TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALEKSI ULKUNIEMI

STAND-ALONE .NET MICROSERVICES WITH CAMERAS AND IMAGE COMPARISON

Master of Science thesis

Examiner: prof. Tommi Mikkonen
Examiner and topic approved by the Faculty Council of the Computing and Electrical Engineering
on 6th May 2015

ABSTRACT

ALEKSI ULKUNIEMI: Stand-Alone .NET Microservices with Cameras and Image Comparison
Tampere University of technology
Master of Science Thesis, 48 pages, 2 Appendix pages
May 2015
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Tommi Mikkonen

Keywords: services, test automation, network, http, image comparison, .ASP NET, PowerShell

This thesis was about implementations of Microservices that uses cameras and image comparison to verify a situation visually. The original solution was done in ad-hoc fashion utilizing existing command-line tooling and it was controlled by dynamic scripting language with managed runtime. The purpose of the original service was to observe quite drastic changes between images taken from the same situation.

There was a need for similar solution in more time-critic environment giving the opportunity to study alternative methods before making decision of upgrading the old solution to be more suitable for the more demanding need. There were some initial knowledge about some libraries that would work in a certain runtime. This gave also an opportunity to develop software using type safe system and remade framework settled for making a new solution. The technology behind both solutions are explained from the significant details. Also the old solution was developed with not so widely known language and its core differentiation features are explained.

This thesis also walks through the old solution from its key concepts offering a baseline for the new solution. After that the new solution is examined from the same key functionality point of view together with some implementation technical notes. The problems that came up during the development and testing phases are looked with the solutions tired and selected together with some documentation about the imaging libraries used.

In this thesis some core differences between the old and new solutions are examined by measurements and observations. Also the bottleneck of the old solution was verified with comparison of different scenarios.

Both of the solutions are used in different environment at the moment. They both expose the same functional interface so replacing the other is possible, maybe even beneficial for the maintenance point of view. Even though the old solution performs well in its current tasks replacing it with the modern one might give that environment more flexibility to extend is test scenario capability.

The notion of synergy between solutions is marked in the thesis and a possibility to compile a library for other one to utilize and gain more performance.

TIIVISTELMÄ

ALEKSI ULKUNIEMI: Mikropalvelut kameroilla ja kuvavertailulla
Tampereen teknillinen yliopisto
Diplomityö, 48 sivua, 2 liitesivua
Marraskuu 2015
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen

Avainsanat: palvelut, kuvavertailu, verkkosovellukset, testiautomaatio, .NET

Tässä työssä tutkittiin laitteen näytöllä olevien tilanteiden tunnistamista kuvavertailua tarjoavan mikropavelun avulla. Alkuperäinen ratkaisu syntyi nopeasti tarpeeseen käyttäen komentorivityökaluja sekä skriptikieltä. Ratkaisun tarkoituksena oli havainnoida varsin huomattava muutos samoista tilanteista otettujen kuvien välillä luotettavasti ja tämän toiminnallisuuden se myös toteuttaa hyvin.

Vastaavalle palvelulle on löytynyt myös muita käyttökohteita, jotka ovat aikariippuvaisempia. Otettaessa huomioon latenssin merkitys palvelusta saatavan hyötyyn nähden, voidaan nopeampaa palvelua soveltaa laajemmalle kohdealueelle.

Näin syntyi tarve joko jatkokehitykselle tai mahdollisuus tutustua uuteen kehitysmalliin. Työssä selvitetään valitun uuden toteutustavan eroja muihin olemassa oleviin malleihin saman ohjelmistokehyksen sisällä. Tämä tarjosi myös mahdollisuuden käyttää vahvasti tyypitettyä kieltä ja helpomman integraation haluttuihin kuvankäsittelykirjastoihin.

Työssä käydään ensin vanhan toteutuksen yksityiskohtia läpi samalla dokumentoiden muutamia työkalukohtaisia ykistyiskohtia. Tämän jälkeen esitellään vaihtoehtoinen toteutus, joka tarjoaa saman rajapinnan käyttäen sisäisiä kirjastoja komentorivityökalujen sijaan. Lisäksi selvitetään uuden ratkaisun toteutuksessa kohdattuja ongelmia ja niihin löytyneitä ratkaisuja, sekä dokumentaatiota kirjastojen käytöstä ja muuttujien käyttäytumisestä. Eri toteutuksia vertailtiin mittaamalla palveluiden vasteaikoja toisiaan vastaavilla testiskenaarioilla ja alkuperäisen toteutuksen pullonkaula paljastui.

Molemmat ratkaisut ovat oikeasti käytössä. Koska kummatkin toteuttavat saman rajapinnan ja toiminnallisuuden, on palvelun korvaaminen toisella mahdollista. Tämä on pelkästään yllläpitosyistä jopa ehkä suotavaa, vaikkakin vanha toteutus tietenkin toimii sen alkuperäisessä tarkoituksessa ihan hyvin. Toisaalta vaihtamalla toteutusta testiympäristöllä on mahdollista laajentaa testien määrää.

Työssä on myös havainnoitu toteutustapojen synergioita ja kylvetty siemen mahdolliselle kirjastojen ristiinkäyttämisen kokeilemiselle tulevaisuudessa.

PREFACE

This Master's thesis was written for Tampere University of Technology

I want to express my sincere gratitude and thanks to Prof. Tommi Mikkonen who has examined my writing trials earlier and has still encouraged me to keep on going and helped me at finding a clearer focus.

Huge thanks goes to my family, friends and colleagues from every organization I've been involved with, for supporting me with good spirit throughout my life!

Aleksi Ulkuniemi

Tampere, 25.11.2015

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Information Interchange
BCP	Best Current Practices
CI	Continuous Integration
CIL	Common Intermediate Language
CLI	Command Line Interface
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLRF	Carriage return followed by line feed
CLS	Common Language Specification
COM	Common Object Model
CPU	Central processing unit
DDP	Distributed Data Protocol
DLL	Dynamic Link Library
DNX	.NET Execution Environment
DNX	.NET Execution Environment
ECMA	European Computer Manufacturers Association
FIFO	First-in-first-out
FS	File System
GC	Garbage Collector
I/O	Input/output
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IIS	Internert Information Services
IP	Internet Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
NIC	Network interface controller
OS	Operating System
PDU	Protocol Data Unit
PS	PowerShell
QoS	Quality of Service
R&D	Research and Development
REST	Representational State Transfer
RPC	Remote Procedure Call
RTT	Round-trip time.
SAP	Service Access Point
SOA	Service Oriented Architecture
STD	Internet Standard
Stderr	Standard Error
Stdin	Standard Input
Stdout	Standard Output
TCP	Transmission Control Protocol
TTL	Time to Live
TTY	Teletype user terminal connected to computer
URI	Uniform Resource Identifier

1. INTRODUCTION

Most of the data available on public Internet used to be first processed by humans. Data published trough web servers were mostly static file shares and pages. Automating various tasks and making life easier have intruded humans since the dawn of times. Humankind have automated the data production, processing and consumption. The life being easy is an open-ended question.

In the field of test automation the requirements are changing constantly as products and product lines evolve. Features are being implemented and they need to be tested. Software industry is not an exception and there is even a phrase to describe yet another framework (YaF) as they have been popping out in recent years. One of the biggest subsection in the field of software engineering is web-development. Good practices tend to survive and make it to the mainstream, sometimes through some extreme.

This thesis is experimenting with a remade framework for network connected programs with command-line interface The ASP.NET 5. The Release Candidate 1 was announced exactly one week before this document is submitted. This means that it is getting official support and more developers are going to try it out. In early 2016 it is planned to be released to manufacturing (RTM). Beta-stage has gone almost a year and the open-source community have been actively contributing to its development in various phases.

Extending existing systems with extra services leads to distributed network of devices that produce data and practices of storing and sharing it. In this type of hypergraph systems there is often a hub that acts as a broker for the data so it can be attached to different services like storages and dashboards. Centralized solutions are working as hubs that everyone has access to. Sometimes utilization of existing infrastructure of other fields of software engineering can be beneficial. In example using cloud storage that has already access control mechanism, back up, distribution and well defined API with existing implementations for almost every programming language.

By default most of the nodes in hypergraph are located as end-points of a cell. This means that most of the applications developed are clients or services. New functionality is added to the system by branching the graph and connecting new nodes. The evolution of networked systems is like a living organism. New connections and nodes pop and unused die. Useful nodes are getting stronger as new nodes are connected to it. For the speed of the evolution creation of nodes and connections should be as effortless as possible.

2. TEST AUTOMATION AND SOLUTIONS

In this chapter an overview to test automation is made. Developing to test automation environment can be quite beneficial for the quality of once own code. Any failure in the tool-chain or in the test itself can result as a test failure and that concerns somebody.

2.1 Test automation solution as a Service

The service oriented mind set has gotten its way ranging from Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) to Operating System as Service, Compiler as a Service and Logging as a Service. The change to view almost everything as a service have made system architecture from integration to service compilation. Libraries are downloaded from repositories as service. Dependencies are called contracts. There are service injections. Software projects can utilize many of these services ranging from cloud hosted servers, platforms for backend and databases to analytics and email-services. Even integration and builds can be done in cloud environment directly triggered by commit to cloud based repository.

The test automation setups and needs for larger companies can also lead to a situation where there are dedicated teams offering the test solutions as a service to feature developers to execute their assets in the systems. Offering up to date result and reports as a service. In the Figure 1 is overview of general purpose test execution framework.

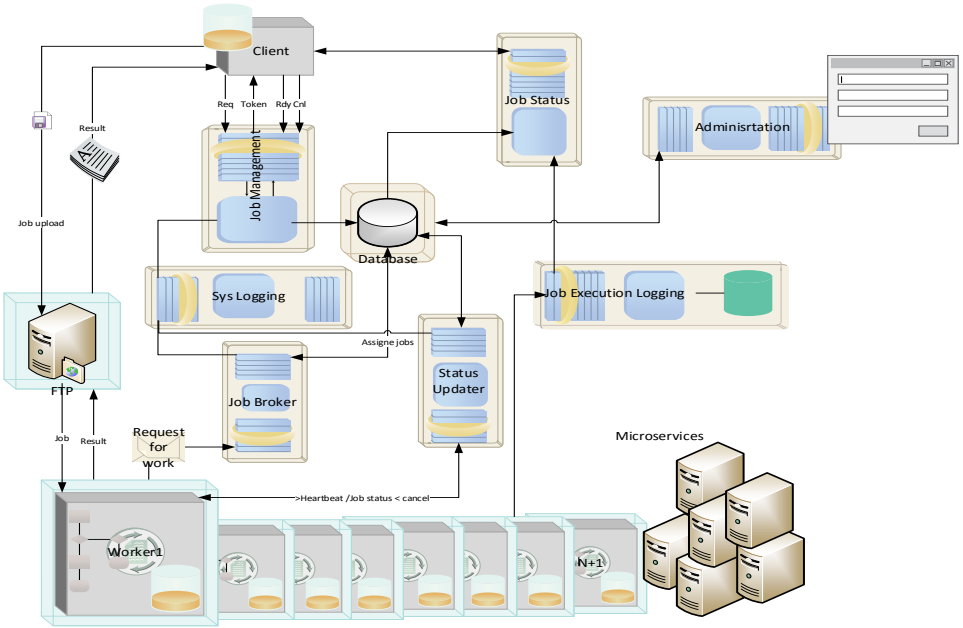

Figure 1. *Service oriented architecture of test platform extended with microservices.*

Development team should be fully responsible of the service they develop in production. This means that software development is not decoupled from operations. “You build it, you run it” noted the Werner Vogels Chief Technology Officer and Vice President of Amazon during the interview. (O'Hanlon, 2006)

Every piece of live code can be tracked to something whether it is a single test run or a part of a suite.

2.2 Special Setups

Time to time need for utilizing external inputs to the systems under test arise. Whether it is sound, light, touch changing the orientation or validating certain situations. For the scope of this thesis a general purpose external service utilizing camera was selected. There was some issues limiting the deployment of the existing ad-hoc solution to perform in required way in another environment.

Flexibility has been a key component for these kind of projects and that is why the reference project was developed with PowerShell which is truly a multipurpose tool. More detailed description of the existing solution is in Chapter 7.

There are parts the testing system where for example the amount of data that test runs generate to different systems can generate pressure for performance. In many cases the speed is not too relevant if there is no direct interface to humans and some later phase data parsing and analyse can take some time. As with any dynamic systems that are heavily utilized the bottlenecks reveal them self.

Anyhow there might be an option for a middle ground between enterprise thigh and coupled to scripting language flexible. Chapter 3 is for short recap on networking. Chapter 4 is for .NET Framework and Chapter 5 for its new modular coming. Chapter 6 opens up the concepts of microservices.

Modularization has been a hot topic for a long time. It is broad concept and can be utilized in many levels of software engineering. One form of modularization that gets attention in this thesis are packets. In the other hand microservices are a typical example of modularity with clear boundaries and dependencies.

3. INTERNET NETWORKING

In this chapter there is short recap of the basic components of Internet networking. Most of the internet applications and services are built upon HTTP-aware server software and intermediaries such as firewalls, load balancers and proxies. Services that are deployed to span multiple hosts need to have a connection to reach one another.

Kurose and Ross are using five layers to describe the networking model in contrast to OSI-models seven. Application, Transport, Network, Data link and Physical. In this chapter most common transport layer protocols and their characteristics are introduced. Addition to that there is a section about latency and throughput.

Encapsulation is the method that nests payload data in to a packet also known as datagram. Every packet has header that holds information for the protocol operating the packets.

3.1 History of Internet

Advanced Research Projects Agency Network (ARPANET) is the seed of the internet as it is today. The project started during the Cold War. It was at first driven by the needs of military. ARPA currently known as DARPA[1] with Defense is a separate agency that answers directly to U.S Department of Defense. It is specialised in highly advanced research. They have funded many research projects in the field of communication and networking.

One turning point was 29th of October 1969 when first message was sent and received via ARPANET. In those four universities ARPANET was deployed. Students and researchers were hacking with them and developing protocols that layered the foundation that rest of us can build and learned from.

At first ARPANET used Network Control Program (NPC) as transport layer protocol achieving host-to-host connection. It was simplex protocol, so that application protocols implanted on top of NPC used two ports in order achieve duplex communication. Uplink and downlink numbered as even and odd. NPC was much later superseded by Transmission Control Protocol (TCP) more of TCP in chapter 3.5.

[1] http://www.darpa.mil/About.aspx

3.1.1 Standardisation

Standardisation work have helped many good solution to be benchmarked and documented to an extent that they have been standardised. This have been crucial in order to achieve interoperability in the field of networking and web technologies.

Early work for internet standards were actual requests for comments (RFC) papers distributed among researchers. They were written in less formal style like todays Internet Draft documents prior being possibly approved as actual RFCs.

RFC are published by Internet Engineering Task Force (IETF) and they are statically serial numbered documents. RFC are categorised by the nature of the document and also by the maturity of the standard or technology it describes. The Process of Internet Standardisation is actually in RFC (2026) form. (Brander, 1996)

The documents that are on the track to become internet standards are categorised as proposed standards, there used to be draft standard in between, but the process have been streamlined by RFC 6410 (Housley, Crocker and Burger, 2011).

Non-standard RFC with experimental category might also be promoted to standard category if they are found to be relevant. Other non-standard RFC category is informational. Informational category is used for other documents such as guidelines and requirements.

Already published RFC can be updated, or made obsolete by submission of another RFC. There are two sub-series that keep currently relevant RFCs linked. Internet Standards (STD) for standard track protocols and Best Current Practices (BCP) for other relevant RFCs. They have their own static numbering, but the RFCs linked to them are changing if needed.

3.1.2 Telnet and A File Transfer Protocol

RFC 1[2] titled Host Software was proposing telnet and requesting for a file transfer protocol that functions with remote host. In April the 1971 Abhay Bhushan published an article titled A File Transfer Protocol (FTP). FTP is the standard that enables users and computer to transfer files from other systems regardless of operating system (OS) or file systems (FS) they are running, assuming they have FTP-server and -client processes running. (Bhushan, 1971)

FTP shares are widely used today as it offers an effective and easily manageable way to achieve cross-platform file exchange. Files are still an important part of computing and major way of storing and distributing data assets.

[2] https://tools.ietf.org/html/rfc1

In the same article he derived computer network usage in to a two broad category direct and indirect. With direct usage he meant the means described in the previous section, such as logged in user with telnet session. The indirect usage he did not just limit for file transfer. He describes what an extended file transfer protocol could do, that facilitate the exchange of programs and data between computers. Usage of other computers storage and even trillion-bit data store computer was mentioned. Having programs in remote host operating on your input and return an output. This is still describing modern computing paradigm very well. (Bhushan, 1971)

Telnet connections and tunnels have enabled service usage over broad range of connections from serial ports through modems and even in the modern Internet. Its successors in form of Secure Shell (SSH) and remote-PowerShell sessions are handy to control or automate tasks on remote hosts.

3.2 Latency and Throughput

Latency is essentially the time taken to send a unit of data from host to host in network. In the context of packet switched networking latency is constrained by networking equipment handling the routes and the physical capabilities of the links transferring the packet from the source host to the destination host.

Latency is a sum of delays that every node and hop in the network contribute. Kurose and Ross distinguish in their book the most important components that together form a total nodal delay. These delays are nodal processing delay, queuing delay, transmission delay and propagation delay. (Kurose and Ross, 2011: 63)

Nodal processing is the time that it takes the router to read the datagrams header information and check if it still has time to live (TTL), determine the route for the packet using the destination IP address and possibly enforce firewall or other rules. Queuing delay comes from the fundamental feature of first-in-first-out (FIFO), assuming no Quality of Service (QoS) or priority rules are applied. This means that the packets have to possibly wait their turn to be transmitted depending of the load of the interface. Transmission delay is the time that it takes network interface controller (NIC) to push the bits of the packet to the link. Transmission delay is the packet length (bits) divided by the link rate (bits/second). Propagation delay depends of the physical medium and the length of the link. The medium dictates the propagation speed, which is close to seed of light in range of $2 \cdot 10^8 \mathrm{m/s}$ to $3 \cdot 10^8 \mathrm{m/s}$ this is usually in the order of milliseconds. (Kurose and Ross, 2011: 63-65)

Looking the latency from the perspective of application to application communication. Latency is then the sum of all the operations that is performed at all the layers of the

model establishing the messaging. In the context of internet trafficking the transport layer overhead can at significance if the message size is small.

Throughput is the amount of data per second that can be transferred. High throughput does not necessary mean low latency. A good analogy for this is to think about a section of a road. Motorcycles and busses are travelling that section, which had the lower latency? Throughput can be increased with parallel links, as throughput can be roughly estimated as instantaneous capacity divided by latency as counting the acknowledgements.

Latency is more important when transmitted data is relatively small compared to bandwidth. Latency can be seen more important as it is harder to negate. If the latency of last link is high there are not much that can be done to fix it. (Cheshire, 1996)

Latencies in traditional networking was an issue building computer clusters from desktop computers. A User-Level Network Interface (U-Net) (Basu et al., 1995) enabled user-level processes to access high-speed communicational devices bypassing kernel mode packet handling and copying to user mode buffer (Basu et al., 1995). Virtual Interface Architecture VIA) is an industry standard that composes many of U-Nets ideas. The problem fixed itself as central processing unit (CPU)

3.3 Internet Protocol

Dynamic nature of internet comes from numerous routing protocols running in the network layer together with the famous IP protocol. Together they are offering best effort packet delivery from host to host in inter connected network. (Kurose and Ross, 2011)

IP-packets also known as datagrams encapsulates the payload that is often itself encapsulated data from one of the transport protocols. Addition to payload these network-layer packet have fields for packet handling in the network. Most important fields are source and destination IP-addresses, length and TTL, so that the packet will not encircle the network forever trying to find path to destination.

3.4 User Datagram Protocol

Used Datagram Protocols (UDP) RFC was written by Jonathan Postel (Postel, 1980). UDP is simple and effective datagram based protocol. It operates without handshake or connection. There is no guarantee for datagram delivery or duplicate protection. UDP operates on top of IP network layer. The UDP-datagram is encapsulated with in single IP-datagram and thus have payload limitation of 65507 bytes with IPv4.

UDP Header is very light with length of 8 octets. The 64 bits are divided for four sixteen bit slots representing optional source port, destination port, length of UDP-packet and checksum for UDP-packet. UDP-packet here refers to combination of header and data.

Data integrity for delivered message is provided with checksum when used value other than zero. If checksums are computed and the result ends to all zeros an arithmetic complement is used instead, that is all ones. (Postel, 1980)

In many situation UDP transmission can be done in the rate of the NIC that is higher than the path rate to receiving host. As there are no cognition control at UDP level transmission should be done at moderate rates not to add cognition to the network. One of the attractive use cases for UDP are tunnelling other higher level protocols establishing point to point connection. UDP-datagrams are then (Fairhurst, 2008)

3.5 Transmission Control Protocol

As stated in the introduction the pathway for network applications in this paper is standard Internet Protocol Suite (TCP/IP). Almost all Application layer protocols that rely on reliable data transfer is built on top TCP which operates on transport-layer.

In TCP sequence number is used for identifying every payload byte. It is incremented accordingly by sender. This allows receiver to reconstruct data from fragments that might be disordered and retransmitted. TCP is a data stream protocol. For the application layer it is TCP is just bytes in stream.

Raw TCP socket can be used to expose tools as a services that operates with standard streams piping standard input (stdin), standard output (stdout) and standard error (stderr) to the socket and forking the processes as TCP sessions are established.

3.5.1 Flow Control and Congestion

TCP controls and adjusts the flow of data segments going from slow start up to Congestion Avoidance phases. Receiver gives hints to sender for adjustments of sliding transmission window. This way there is no excessive segment loss due receiver's buffer being full. TCP protocol acknowledges (ACK) received bytes of data by sending cumulative byte counter with responses. Selective Acknowledgement (SACK), when supported, allows fast transfer for streams and data spread over multiple segments as it allows holes in the stream that is acknowledged. This also benefits throughput in high latency networks.

Fast retransmits happens when segment loss occurs based on estimation of RTT of the connection and missing segments ACK.

3.5.2 Three-way handshake

The reliability of TCP comes with a cost of a relatively expensive three-way handshake.

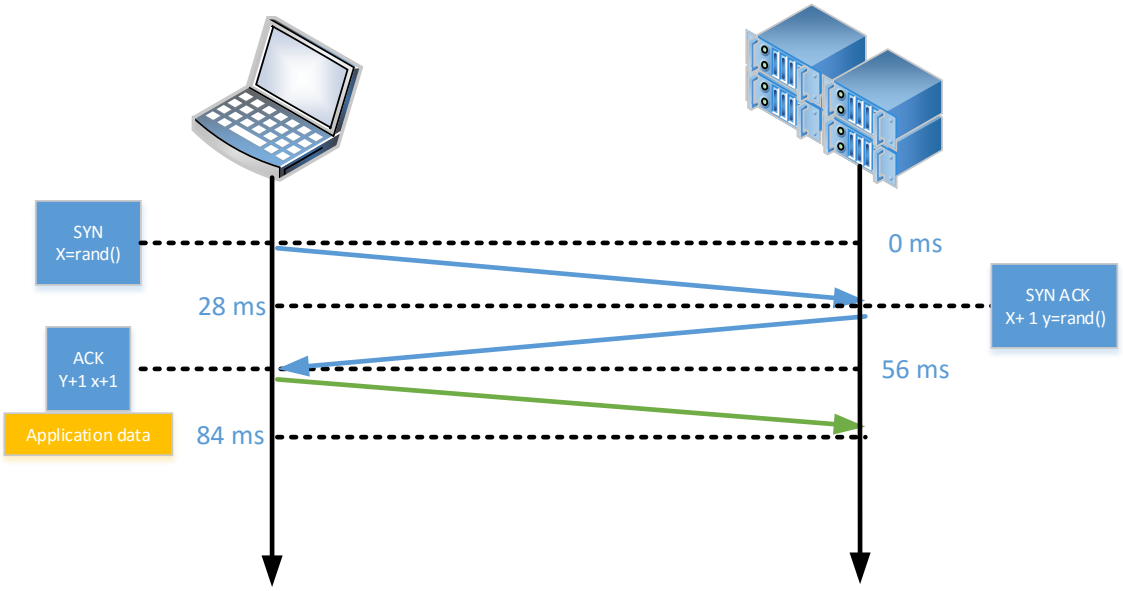

Figure 2. *Three-way handshake after Grigorik (Grigorik, 2013: 15)*

TCP handshake time at Google servers with thumb images and map tiles were over 20% of the total HTTP latency in June 2011 (Radhakrishnan et al., 2011). This drove the development of TCP Fast Open (TFO) an experimental update to TCP now in form of RFC (Cheng et al., 2014). Support for the TFO protocol extension was merged into Linux kernel mainline in version 3.16.

TFO needs client and server support in order to function. There are Message Authentication Cookie (MAC) used to bind the IP-address of the client to the requests for TFO in the SYN message.

First TCP connection is regular 3WH where supported client sets Fast Open option and an empty cookie field in the SYN. Server then generates a unique cookie for the client and sends it back with the ACK. Next time the client initiates TCP connection with the server it can use the cookie to verify its identity and the server can send payload with the SYN response.

3.6 Hyper Text Protocol

The Hypertext Transfer Protocol (HTTP) most common version 1.1 is defined and being standardized in Internet Engineering Task Force (IETF) Request for Comments (RFC) indexed 2616. (Fielding et al., 1999)

Considering the maturity and wide spread usage of the HTTP this section will not discuss the implementations of the protocol itself rather the characteristics and some techniques used to bend it to perform in different manner than the original use cases were.

One of the fundamental behaviour of HTTP protocol is that it is used on demand from the client point of view.

3.6.1 Connection

HTTP session is formed between two processes that can be running in different hosts. It is client-server oriented. In this chapter the process initiating the connection and sending requests is called client and the recipient is called server and it forms sends the response.

HTTP session is established by a client opening a TCP connection to a HTTP port 80 or 443 in server. Underlying TCP provides reliable data transfer for HTTP messages.

3.6.2 Persistent Connections

Often in HTTP API calls in the TCP flow is just few round trips. Requests and Responses should be reasonably small. For bigger data queries responses should be divided into multiple requests with usage of *Range* fields in headers. In practice the optimal query size may vary according to the client capability to handle the data set. For browser operating on JSON

There are possibility to perform multiple HTTP request – response pairs using single TCP connection sequentially. This persistence of connection is also known as *keep-alive.*

Keep-alive value is set in the request header connection field option in HTTP/1.0 to inform the recipient that client supports and wishes to use this feature. Server sets *keep-alive* to the response if it wishes to support the feature. HTTP/1.1 or later versions use persistent connections by default. This means that the connection is persistent and no need to set connection option in header.[3]

Persistent connections can be used until one of the recipients sets the connection option to *close* or with HTTP/1.0 lack of *keep-alive* connection option in response is interprets that connection is non-persistent.

[3] http://tools.ietf.org/html/rfc7230#section-6.3

3.6.3 Uniform Resource Identifiers

Uniform Resource Identifier (URI) is a string notation used to identify resources. There is an RFC 2396[4] that specifies the generic syntax of the string. There are limitation regarding character set used and thus escaping is needed to encode reserved or out of US-American Standard Code for Information Interchange (ASCII) characters. Space (' ') for an example is escaped as "%20" or in some implementations '+'. The URI syntax is shown in the figure 3.

```
URI = scheme:[//[user:password@]host[:port]][/]path[?Query]][#fragment]
Unreserved = 'US-ASCII'| mark
Mark = "-" |"_" |"." |"!" |"~" |"*" |"'" |"(" |")"
Reserved = ";" |"/" |"?" |":" |"@" |"&" |"=" |"+" |"$" |","
Escaped = "%" Hex Hex
Hex = Digit |"A" |"B" |"C" |"D" |"E" |"F" |"a" |"b" |"c" |"d" |"e" |"f"
Digit = "0" |"1" |"2" |"3" |"4" |"5" |"6" |"7" |"8" |"9"
```

Program 1. *Uniform Resource Identifier Syntax after RFC 2396[4] and Wikipedia[5]*

The scheme component is case-insensitive for e.g. HTTP and http are resolved the same. Some outside boundaries can limit the length of the fields. Domain Name System (DNS) is often used to resolve the host marked as labels divided by dots. These DNS labels can be up to 63 characters long and restriction for the whole string is 255 8-bit octets. [6]

Subsets of URI exists and the most common one is Uniform Resource Locator (URL). URL is used to identify resources by network location together with representation scheme often named after some protocol that is involved in the process. URLs are part of the HTTP API of the services introduced in chapter 7 and 8 in Figure 4 there is an example of an URL that with the http-protocol get-request would set camera indexed as 0 properties.

```
"http://localhost:5000/api/camera/0/SetCameraControlPropeties?Tilt=none&Zoom=2"
```

Program 2. *Example Uniform Resource Locator*

3.6.4 Messages in HTTP

Messages in HTTP/1.1 are either request or response type, these messages are in the generic ASCII message format described in RFC 822[7]. Each message consist of header fields and optionally a body separated by null line which essentially itself is carriage return followed by line is feed (CRLF). According to (Fielding et al., 1999), all field names in request and responses are case-insensitive.

[4] http://tools.ietf.org/html/rfc2396
[5] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[6] https://tools.ietf.org/html/rfc1123#section-6.1.3.5
[7] http://tools.ietf.org/html/rfc822

Request message starts with request-line defining the method, resource and protocol followed by headers each in separate line. Optional body with Content-Length other than zero or Transfer-Encoding as chunked together with Content-Type and encoding is important metadata when parsing the body.

Request headers: Accept, Accept-Charset, Accept-Encoding, Accept-Language, Authorization, Expect, From, Host, If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, Max-Forwards, Proxy-Authorization, Range, Referer, TE and User-Agent.

Response header are Accept-Ranges, Age, ETag, Location, Proxy-Authenticate, Retry-After, Server, Vary, WWW-Authenticate.

General headers that can be either in request or in response are Cache-Control, Connection, Date, Pragma, Trailer, Transfer-Encoding, Upgrade, Via and Warning.

HTTP/1/1 Request methods GET and HEAD must be supported by servers. Other optional methods are OPTIONS, POST, PUT, DELETE, TRACE and CONNECT.

GET method is used to retrieve any information URI points to. It can be a output of a process or a static file. There are options to produce conditions using the request header fields that starts with “If-” e.g. If-Modified-Since. Partial retrieval uses range-header field.

HEAD is identical to GET without the body in the response.

OPTIONS is used to request available methods on the target resource. Response could have Allow field e.g. "Allow: GET, HEAD".

POST method is used to send data entity towards the resource. It might get appended to data set or used as a method of submission of a form.

PUT indicates that the payload should be stored under the Request-URI. This can lead to modification (over-writing) or creation of new resource.

TRACE should be reflected the entire http-message back with status-line "200 OK" as Content-Type of "message/http". This can be useful for debugging how the server actually receives the message and its content, however it is really not used that often and with ever complicated network setups and added headers it should be turned off as it exposing vulnerabilities. (Grossman, 2003)

CONNECT is used to turn the opened socket to be a tunnel to the endpoint requested. This is used to open connections through proxies and also to enable SSL tunneling.

In Program 3 we have an example of http-get request send from JMeter[8] to the service described more detail in chapter 8. The packet was captured using Fiddler[9] as a proxy. Fiddler can be useful tool while debugging HTTP related systems.

```
GET http://localhost:5000/api/camera/0/compare/test1 HTTP/1.1
Connection: keep-alive
Host: localhost:5000
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.2.6 (java 1.5)
```

Program 3. *Example Request Message from JMeter client*

Response messages status-line has the protocol, status code and a reason phrase. The status code and reason phrase form a pair, code for machine and phrase for human users. There are 5 categories for status-codes which are 3-digit unsigned integers.

1xx: Informational, 2xx Success, 3xx Redirection, 4xx: Client Error and 5xx: Server Error

Most common response code should be "200 OK" and most memorable for common users might be "404 Not Found". "405 Method Not Allowed" and "501 Not Implemented". When service fails to comply with the request an appropriate response should be send as response.

```
HTTP/1.1 200 OK
Date: Mon, 23 Nov 2015 21:39:19 GMT
Content-Type: application/json; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked

3a
{
  "msg": "result",
  "id": "967",
  "result": 613
}
0
```

Figure 3. *Example Response Message from Kestrel*

3.6.5 Clients

The term Ajax was first introduced to world by Jesse James Garret from adaptive path. The name comes from the terms Asynchronous JavaScript and XML.[10] Evolution of XMLHttpRequest (XHR) in JavaScript started when engineers in Microsoft working on Outlook web application developed an asynchronous ActiveX component to fetch HTTP data. They shipped it with Internet Explorer in MSXML library. It was only because they

[8] http://jmeter.apache.org/
[9] http://www.telerik.com/fiddler
[10] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/

had contact to that team and due to Explorers tight shipping schedule. Component was named *Microsoft.XMLHTTP* even though the component didn't have much to do with XML.[11]

3.7 Techniques for Server push

The limitations and restrictions of browsers communicating with web servers with HTTP have driven some creative usage of http protocol. There is no direct way for server to initiate communications with client.

3.7.1 Polling

Simplest solution to keep client up to date with the server is for client initiating a request to the server and analyze the response in some interval. Polling is in line with the original concept of HTTP's restful request-response architecture. Polling will naturally increase the number of request to the server, but even the traditional webservers could be used to scale this approach.

Polling can be made less resource consuming depending on context. If the data is not prone to change often utilizing a HEAD request can save some server resources as the response holds just headers without a body. The *LastModified* field can then be used to determinate if the resource have changed. Some application has such execution phases that allows implementation of lazy polling. In lazy polling the resources are not polled until they are needed or determinate invalid. Possible scenarios for lazy polling would be if the client is service itself and it is not serving clients or if the data is freshly cached. Another one would be a user application that is inactive and there is no constraints for updating the interface or data for notifications.

3.7.2 Long Polling

Long Polling is a variant of polling 3.7.1. The main difference comes from server implementation. When server receives request it will wait with the respond until it has an update. The client will always have an asynchronous request pending for server response. The connection might be dropped by intermediates due inactivity, some implementations of HTTP request with keep-alive do send TCP level keep-alive packets to refresh the connection. Requesting client libraries have also implemented timeout values for HTTP requests, in case the server is malfunctioning. This means that new request need to be initiated also in case of disconnection or timeout.

[11] http://www.alexhopmann.com/xmlhttp.htm

With Long Polling server can deliver data to the client at any time. This is significantly reducing latency. Alex Russell (Russell, 2006) coined the term Comet when blogging about this event-driven, server-push data streaming style.

Handling a requests that take long time to complete can be very resource hungry for the webserver that uses blocking threads for the implementation. Every client have an open connection waiting the event at server to trigger the response.

The HTTP/1.1 introduced data transfer mechanism where the body of the response can be send as series of chunks. It reduces page load times with dynamic content as serving the page can be started without knowing the content length. It also allows server to stream content.

The header field Content-Length is omitted and use Transfer-Encoding set to "chunked" instead. The end of the body is indicated with null line (CRLF).

Forever frame also known as hidden iFrame technique is considered to be a clever hack where a chunked transfer of an object is left open pushing <script> tagged pieces of code to be executed on browser.

HTML5 Server-Sent Events In HTML5 a simple protocol for chunked messages was established to enable server send messages in long lived HTTP-session.

3.8 Websocket

The development of WebSockets were driven by the web applications community to assert the issue of lacking protocol for bi-directional communication between the browser and the server. As more web applications started to implement nearly real-time functionality with server push techniques.

Websocket is implemented by upgrading HTTP connection and utilizing the undelaying TCP socket for lightweight bi-directional messaging. This allows developers or libraries to implement event based listeners for the WebSocket protocol that fires on connection, message and close events.

There are several implementations that offer solutions for basic use cases like SignalR[12] or SockJS[13].

[12] https://github.com/SignalR/SignalR
[13] https://github.com/sockjs

4. .NET FRAMEWORK

In this chapter some fundamentals of the .NET Framework is explained that the .NET Core Project it in chapter 5 is easier to comprehend. This is also the part of the stack behind the service introduced in more detail in chapter 7.

All .NET Frameworks in desktop and server includes runtime described in subsection 4.2 and base class libraries that other managed code can depend and derive from (4.3) together with some managed libraries. They are distributed as packages that are in place updates to earlier versions. Updating to newer version of .NET in a machine can break existing applications. The backward compatibility is introducing limitation on adapting innovations in development and delaying releases of the framework. (Landwerth, 2014)

4.1 Overview

The full .NET Framework is operating at machine-wide level. It has made is mark mainly as Windows only framework provided by Microsoft to offer developers unified tools and libraries to target its platform. Most common projects that utilize framework are Windows Presentation Framework (WPF), Windows Forms and the main web stack ASP.NET often on top of Internet Information Services (IIS).

In subsection 4.1.2 the concept of Common Type System that is used in the CIL is explained. In figure 7 an overview of some managed projects that targets .NET runtimes and libraries is presented.

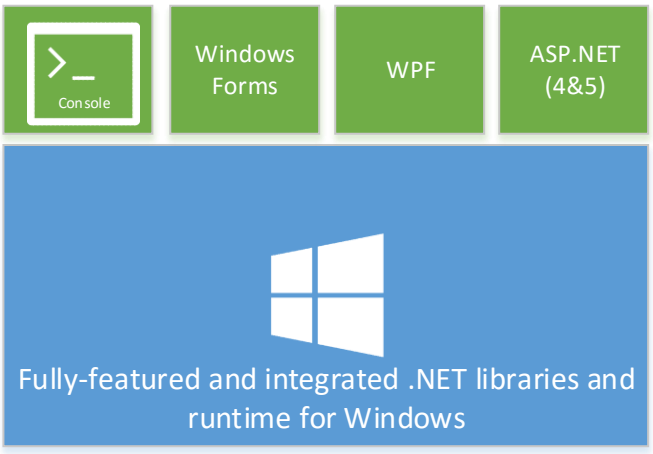

Figure 4. *Centralised .NET Framework for Windows after .net foundation*[14]

[14] https://dotnet.github.io/about/overview.html

4.1.1 Common Language Infrastructure

Common Language Infrastructure (CLI) refers to the language-neutral platform that is defined in ECMA-335[15] standard. The main purpose is to compile high-level languages to Common Intermediate Language (CIL) which has definitive instruction set. CIL is assembled and then run in virtualised environment that compiles CIL to platform- and processor specific machine code in runtime using just in time compilation (JIT). In Figure 8 an overview of the CIL infrastructure is visualised with high abstraction. At top there are code in human readable forms.

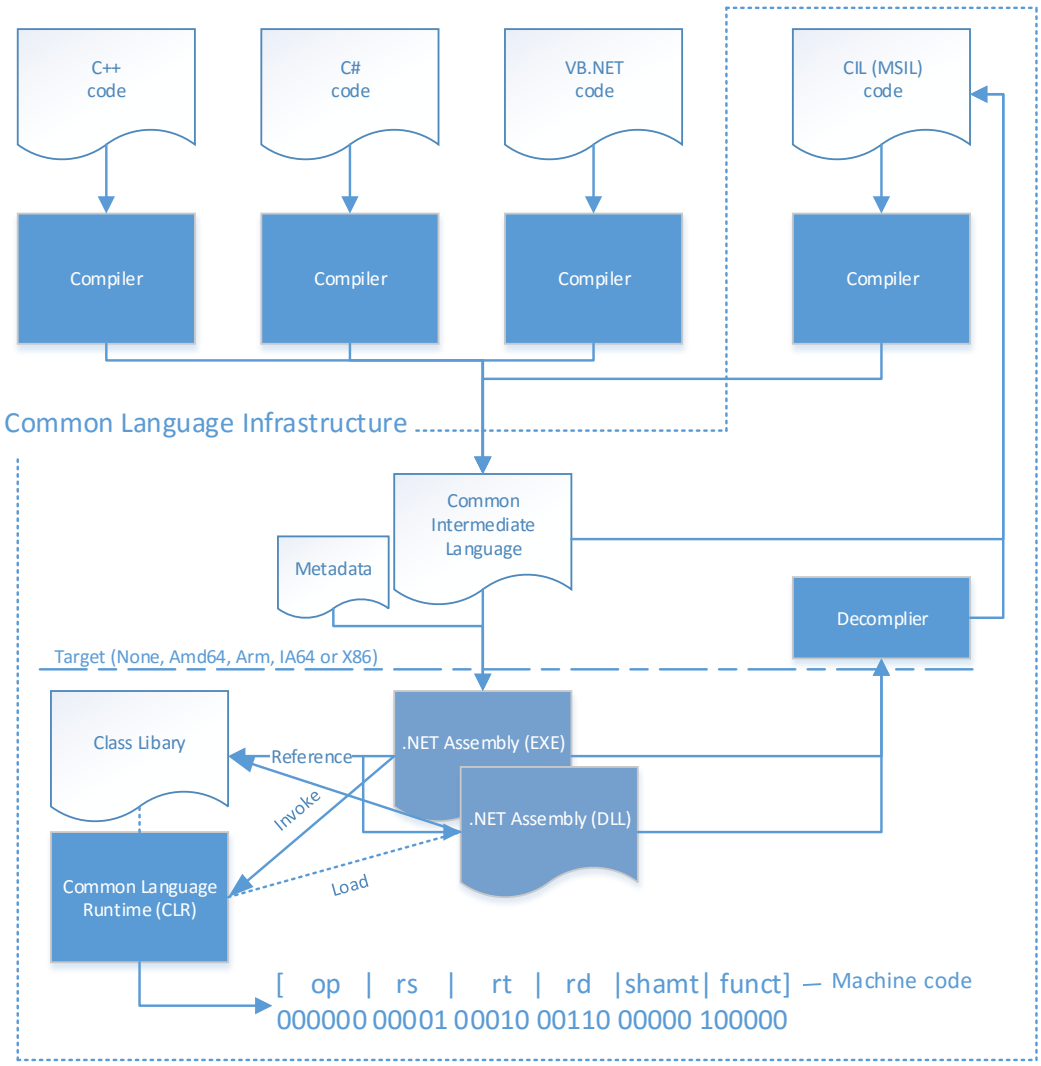

Figure 5. *Common Language Infrastructure overview in .NET Framework*

Assemblies that are directly targeting CLR are called managed, this has the benefits of common exception handling, component interaction and debugging services. Managed

[15] http://www.ecma-international.org/publications/standards/Ecma-335.htm

code can integrate to other managed code with the simplified model regardless of the original language, as it is compiled to CIL and essentially run in same virtual environment. It is possible to mix managed and unmanaged code. Using unmanaged code from managed is done through interoperability interfaces. Utilizing managed assemblies in unmanaged code in another hand is achieved either loading the managed assembly with Common Object Model (COM) or bootstrapping a CLR for loading the assemblies.

4.1.2 Common Type System

Common Type System is designed as a super set of Common Language Specification (CLS) which sets boundaries and limitation for the source code ensuring it to fully interact with other objects regardless of the language it was written. Cross-language interoperability is achieved in .NET by settings the rules for compliers and developers targeting the system. Main purpose of CLS is to assure type safety and code verifiability. CLS was designed so that most languages can support it. (msdn, 2012)

A type in the .NET Frameworks managed code is either value or reference type.

Reference typed objects are storing a fixed size references pointing to the actual value stored in the heap. When assignment of a reference type is made to variable a copy of the reference pointing to the original instance value is assigned. If a new object that is an actual copy of the referenced object is needed it can be achieved if the referenced object has implementation of *Clone* method.

An object with value type is allocating the actual data presenting the value. Sizes of value types are known compilation time. Assigning value type to variable results the actual value to be copied to the variable in stack. Most primitive data type's compliers use are value types in .NET excluding for example *String* which is actually immutable reference-type.

Immutable types have value type like behaviour because all the methods that would result a state change are returning a new object of the immutable type instead (Hartl, 2010).

Types are further divided into following five categories classes, structures, enumerations, interfaces and delegates.

All classes are implicitly derived from top level *System.Object* and can also be derived directly from other class. Like in many object-oriented systems class defines the operations and data that is

4.2 Common Language Runtime

Common Language Runtime (CLR) is the .NET Frameworks environment that actually runs the code and offers services for development and debugging. CLR is virtual execution system. CLR relies on the metadata saved to common language portable executables (PE) at compilation time.

4.2.1 Start-up

In case of Windows the operating system (OS) detects managed code when image is being loaded it then loads native MsCorEE.dll and calls its function _CorValidateImage[16]. Image is loaded in to memory and if confirmation for validity is passed an entry point for managed code in memory is added to image header. After validation the execution comes back to operating system image loader with status if the loading was successful with executable assembly _CorExeMain- or in case of dynamic link library (DLL) _CorDllMain-function is called regardless what the actual entry point in the image was. In pre XP there was a bootstrap in the managed image to do indirect call. These CorNxxMain functions initializes the CLR, locates the managed entry point from the image header and begins execution. When CLR ends the function returns the return code of the manage code int32 for the operating system.

4.2.2 Assembly management

At the runtime CLR checks what class libraries to load and what version of .NET the PE was built against. CLR then generates the CPU specific code from the CIL inside the manage-code using just in time compilation (JIT), during this phase it also verifies that the code and its metadata for type safety. JIT is done by first individual method invocation bases for that method. JIT compilation results are bound to the CLR process doing them and are not being shared across with other processes.

Native Image Generator (NGEN) performs ahead-of-time compilation and it is used to compile whole managed-libraries and save them to Global in Native Image Cache where CLRs find them. This enables same compilation result be used by more processes if they share the dependency.

Global Assembly Cache (GAC) is a place where applications can install their strongly named and signed libraries to be shared with others. .NET Framework uses this place for its core component libraries. PowerShell installs its System.Management.Automation.dll there too to be discovered with processes that want to utilize it.

[16] https://msdn.microsoft.com/en-us/library/4ce9k7xb(v=vs.110).aspx

4.2.3 Memory Management and Garbage Collection

CLR is also responsible of the memory management for managed code. In 32-bit Windows each process has 2-GB of virtual address space in user-mode. The address space can get fragmented and process can run out of memory when requesting memory big enough block if there is no free block to be reserved.

CLR initializes Garbage collector (GC) for each process and GC allocates memory for managed applications from the heap with Win32 VirtualAlloc function. It can also allocate new segments on need bases and releases them to the operating systems when they are cleared from objects.

GC performs cleaning when GC.Collect method is called directly, although not advised if not unique situation, or when memory certain conditions are met.

The first condition is that the system has low physical memory despite having virtual memory free for the process. This reduces paging in OS memory management.

GC also watches the process heap size continuously in certain intervals at runtime and adjusts to the acceptable level by performing.

The clearing process is simplified as walking the tree –algorithm starting from the root object and marking objects as referenced or not. Not referenced are considered dead. There is also compacting phase that live objects are being moved to near each other and dead space freed making the heap size smaller. To reduce fragmentation and to make compacting easier larger objects and smaller objects are also in moved to separate segment.

Objects that are dead are often recently created to make use of this notion of long-life and shot-life objects GC also marks objects into three generations. Objects starts from the generation 0 and object that survive collection are moved to next generation. Using generations means that the collection phase can be targeted to certain generation. Generation 2 objects can be moved to older segments in allocated memory and collecting long-life objects together.

There are two implementation for GC one for workstations keeping desktop applications responsive and running the GC only on one thread and concurrently letting CLR utilize the others. Server version in another hand is tuned so that each CPU core has its own memory section allocated from the heap and GC is run simultaneously on all sections in separate threads. (Meier et al., 2004)

4.3 Base Class Libraries

Base Class Libraries (BCL) in .NET Framework are offering the essential classes for building applications for .NET. These are available in the system that has .NET Framework installed.

Few classes under Microsoft offer support for Microsoft tooling such as MSBuild and some scripting and programming languages (CSharp, JScript, VisualBasic, VisualC) that have support for compilation and code generation.

Most of BCL classes starts with name System. In the root of System there are the fundamental base classes that define the most uses cases ranging from basic data types such as enumerations through exceptions to interfaces and events. In the child namespace there are System.Collections and System.IO and many other useful libraries.

For networking there are two libraries. System.Net containing common protocols and socket implementations. System.Web offering types for browser server communication. Both of these libraries were used in the PowerShell implementation in chapter 7.

5. .NET CORE

This chapter provides information about the restructuring that .NET Ecosystem is undergoing together with the Windows platform and how it is at the moment. The .NET has divided during its lifespan into separate platforms for supporting desktop, phone, store and server -applications. This factoring has led to problems when targeting multiple .NET platforms within single project which have been asserted earlier with portable class libraries, unified API shapes and shared projects. (Landwerth, 2014) (Saxena, 2014)

In the process of making .NET cross-platform compliant to simplify cloud deployment of its technology Microsoft released the .NET Core (DNC) as collection of an open source projects. Section 5.1 explains the differentiation between DNC and .NET Framework. Section 5.2 is introducing the main repositories used to develop .NET core and their purposes. Section 5.4 explains some details about the packet management solution currently at use. Section 5.5 introduces the modern compliers and their functionalities and common use cases.

5.1 Differentiation to .NET Framework

In contrast to the .NET Framework, described in chapter 4, DNC is not centralized all-in-one installment of the framework, but a self-contained modular implementation of .NET. DNC is utilized as packets and shipped together with every application. This allows developers to fully manage the environment and dependencies the application is run against without concerns about centralized .NET Framework installment and versions.

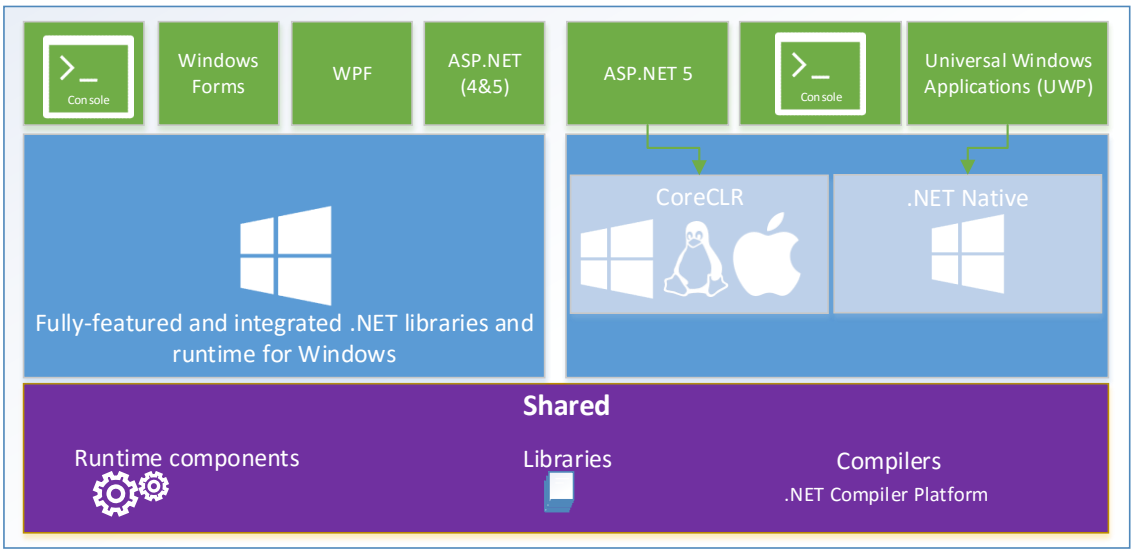

Figure 6. *Overview of current .NET ecosystem after .net Foundation*[14]

5.2 Open Source

DNC lives in GitHub repositories. There are one repository called dotnet/core [17] functioning as starting point and to distribute release notes about the whole .NET Core system.

The dotnet/CoreFX[18] hosts the frameworks implementations for the core classes. To boost the platform agnostic implementations of libraries a reference sources[19] of .NET Framework are published to another read-only repository.

The.NET Core runtime is in repository dotnet/CoreCLR. This holds the base library *mscorelib* that implements GC, JIT compiler and the basic types of Common Type System together with some low-level classes that are platform depended.

Alternative .NET Core runtime solution optimising for ahead of time compilations (AOT) scenarios is located at dotnet/coreRT[20].

For the scope of this thesis and for the solution in chapter 8 ASP.NET[21] repositories are a significant source of packets and information.

5.3 Runtimes

It is clear that managed code needs to be hosted or compiled into native code before it can be executed.

A native execution environment that bootstraps the CLR and runs managed code, similar what OS does in case of execution of managed assembly as described in previous chapter exists in aspnet/dnx repository together with some utility libraries for compilation and packet management.

A NET Version Manager DNVM[22] is a command-line tool for developers to manage DNX versions. At development time developer can use this tool to install or change the dnx bin-path in PATH variable. Every different DNX version installed has its own folder inside "%USERPROFILE%\.dnx\runtimes\." Internally DNVM is a PowerShell script dnvm.ps1 that is bootstrapped with dnvm.cmd file that is listed in PATH. Changes of DNVM done in PowerShell session are stored to "%USERPROFILE%\.dnx\temp-set-envvars.cmd" and the cmd.exe executes and deletes that after the PowerShell returns.

The main program to is however the dnx executable. It is a command-line application that accepts arguments shown in Program 4. It is used for hosting the applications. It can be

[17] https://github.com/dotnet/core
[18] https://github.com/dotnet/corefx
[19] https://github.com/Microsoft/referencesource
[20] https://github.com/dotnet/corert
[21] https://github.com/aspnet/home
[22] https://github.com/aspnet/dnvm

used directly to DLLs or to target folders with project.json file and to run certain a command specified in the project file. From the arguments in the Program 4 output we see that it also offers hooks for debugger mainly for Visual Studio or WinDbg[23].

The project own source code (not the dependencies) in the project is compiled by DNX at runtime this makes development with this setup more like scripting. A setup with PowerShell script or other external[24] means to start the dnx when it exits and give dxn –watch flag so it exists itself on file change.

```
--project|-p <PATH>     Path to the project.json file or the application
                        folder. Defaults to the current folder if not provided.
--appbase <PATH>        Application base directory path
--lib <LIB_PATHS>       Paths used for library look-up
--debug                 Waits for the debugger to attach before beginning
                        execution.
--bootstrapper-debug     Waits for the debugger to attach before bootstrapping
                        runtime.
--framework <FRAMEWORK_ID>Set the framework version to use when running
                        (i.e. dnx451, dnx452, dnx46, ...)
-?|-h|--help            Show help information
--version               Show version information
--watch                 Watch file changes
--packages <PACKAGE_DIR>  Directory containing packages
--configuration <CONFIGURATION>  The configuration to run under
--port <PORT>           The port to the compilation server
```

Program 4. *Listing of dnx commands.*

DNX is also used to bootstrap DNX Utility[25] (DNU) with dnu.cmd in similar fashion as PowerShell was used with the DNVM. DNU is actually Microsoft.Dnx.Tooling.dll with program class initializing command-line utility. DNU handles project dependencies relying on information in project.json. In Program 5 we have listing of dnu commands.

```
build             Produce assemblies for the project in given directory
                  clear-http-cache  Clears the package cache.
commands          Commands related to managing application commands (install,
                  uninstall)
feeds             Commands related to managing package feeds currently in use
install           Install the given dependency
list              Print the dependencies of a given project
pack              Build NuGet packages for the project in given directory
packages          Commands related to managing local and remote packages
                  folders
publish           Publish application for deployment
restore           Restore packages
wrap              Wrap a csproj/assembly into a project.json, which can be
                  referenced by project.json files
```

Program 5. *Listing of dnu commands.*

[23] https://msdn.microsoft.com/en-us/library/windows/hardware/hh406274(v=vs.85).aspx
[24] https://github.com/aspnet/dnx-watch
[25] https://github.com/aspnet/Home/wiki/DNX-utility

5.4 Packet Management

DNC utilizes at the moment just NuGet to distribute packages. NuGet is known among Visual Studio users as the default package manager used to consume packages. NuGet repositories are centralized and referred as feeds. In Program 6 are the default repositories used in the development machine.

```
Default Stable: https://www.nuget.org/api/v2
Default Unstable: https://www.myget.org/F/aspnetvnext/api/v2
```

Program 6. *Listing of used NuGet feeds from DNVM console.*

Developers are not tight on using Visual Studio or any other integrated development environment (IDE) for utilizing packets in their projects. Dependencies to packages and their versions are stored in the project.json file in the root of project folder together with other relevant details about the project. As one can see from the file-extension that JSON (JavaScript Object Notation) is replacing XML as project file format. Once the dependencies are modified DNU can be invoked to restore and build them to cache.

Restore phase of DNU actually downloads the whole graph of dependencies so the project.json files needs to list explicitly just the top level packets needed. Internally DNU creates project.lock.json file next to the real project file where all the dependencies are.

5.5 Compilers

NDX uses the latest RyuJIT Complier[26] to compile 64 bit code. RyuJIT shares codebase with JITx86. JIT64 was slow to execute with earlier implementations as they were based on the C++ optimizing compiler and with emphasis more on code quality than compilation throughput as 64 bit was used mostly on servers.

The other compiler used by DNX to compile the .cs files at start-up time is in a library Microsoft.Net.Compilers. Codename "Roslyn" is a Compiler as a Service offering set of APIs for code and compilation analysis through Microsoft.CodeAnalysis packet. It is the default compiler for .NET in the latest version. In Roslyn the compilation pipeline is componentized offering information and services about parsers syntax tree, symbols used, binds made and code emitted. Utilizing this information a more in depth knowledge of project code base can be offered. Managed code related tools and application can be built against these libraries. (Kean, 2015)

There is also .NET Native compiler used mostly to release for windows 10. For example universal application packets use ready to run precompiled binaries for the platforms.

[26] https://github.com/dotnet/coreclr/blob/master/Documentation/botr/ryujit-overview.md

6. MICROSERVICE ARHITECTURE

Architecture in context of software is used to describe the abstraction of elements in system and their relations to each other in runtime during different phases of execution and deployment. The components forming a system have their internal architecture and as such, complex systems have multiple layers of architecture that describes the behaviour on that level. Architecture level recursion can be run down to the most basic system elements that cannot be broken into less abstract form. (Fielding, 2000: 5-7)

The term “Microservice Architecture” describes a way that applications are formed from suites of independently deployable services. Even though micro is used in the name of the style, it does not limit the actual size of the service provided. Microservices can themselves can be combined and utilised to form more complex services and applications. (Lewis and Fowler, 2014)

Good software architecture and its design decisions should be made within the context of the requirements of the system under design. Such requirements can be social, behavioural and functional. Fielding notes that Design-by-buzzword is a common occurrence and some of such behaviour occurs because of the reasoning behind a good software architectures is not clear when those architectures are selected for reuse. (Fielding, 2000)

In this chapter the microservice architecture is looked through the basic software element types used by Roy Thomas Fielding in his dissertation Architectural Styles and the Design of Network-based Software Architectures. First a detour to Unix-philosophy as it have an influence to the simplicity and isolation that microservices seek.

6.1 Background - Telnet

Steve Crocker outlined Telnet session protocol in the first RFC titled Host Software the simple usage of the network from Network Working Group (NWG) that could be adapted by wide class of users. It seemed natural to provide the ability to use remote host as if it had been dialled up from TTY (Teletype) terminal. The round-trip time (RTT) of half a second at that time was seen as a problem for the remote console. (Crocker, 1969)

In February the first cut of the proposed telnet protocol was published as RFC 97. Telnet is process run in user’s local host that allows them to connect to remote host and give TTY like terminal to gain interactive service over ARPA network. Telnet is cross-platform as it utilizes standard markup and characters conventions in its transfers. Introducing echo mode to allow input be echoed at user agent level gaining more responsive console experience. (Melvin and Watson, 1971)

6.2 Unix-philosophy

Many people from Unix-community have contributed to the philosophy that can be defined as Unix-philosophy. In this chapter three key points, picked by early author of the topic, are cited as they are complementing to the idea behind microservices.

Malcom Douglas McIlroy (McIlroy, Pinson and Tague, 1978), the inventor of pipes, wrote about the characteristic style of Unix-systems while working in Bell Laboratories. The famous maxim of "make each program to do one thing well. To do new job, build afresh rather than complicate old programs by adding new features".

The rule of silence can be drawn from the article as well: "Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input". (McIlroy, Pinson and Tague, 1978)

Also notion complementing agile software development: "Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them". (McIlroy, Pinson and Tague, 1978)

6.3 Componentisation

Components are one of the cornerstones of many architecture models. Reusable common libraries have become part of the most programming languages and paradigms. This have made development and reusing implementations of complex functionality easier.

Lewis and Fowler (Lewis and Fowler, 2014) defines components in microservice architecture as "units of software that are independently replaceable and upgradeable". This means that components are essentially deployed as independent services.

Microservices often use libraries internally in their implementations. With libraries defined as components that are loaded and called in-memory in contrast to services that communicate with out-of-process mechanisms such as web service requests and remote procedure calls (RPC). (Lewis and Fowler, 2014)

Service components have to define their interface in explicit way. As services are themselves defined by their interface. When the application programmable interface (API) is based on common protocol and message conventions the actual implementation of service component is language-agnostic. Constraints to language selection that implementation for the interface is doable in chosen language.

There are a rule of three related to software reuse. Robert L. Grass (Glass, 2003) originally pointing out as rule of thumb that “reusable components are three times as hard to build and should be tried in three different settings”. The rule can be extended to API’s which should be used with three different clients before they find their form.

6.4 Connectors

Fielding (Fielding, 2000: 10) summarizes that: “Connectors enable communication between components by transferring data elements from one interface to another without changing the data”. In architecture diagrams the connectors are often just lines between components. Internally they can do data transformations and content encoding but they are reversed on receiving end.

Shaw and Clements in their paper about architectural styles classification (Shaw and Paul, 1997) defines connector as mechanism that meditates communication, coordination, or cooperation among components. Included examples shared of their observations were representations, remote procedure calls (RPC), message-passing protocols, data streams and transaction streams.

In the context microservice architecture the connectors are kept very simple utilizing the existing networking mechanisms. Usually exposing an API through networking socket defined by uniform resource identifier (URI) these are also known as Service Access Points (SAP).

Perry and Wolf (Wolf and Perry, 1992) wrote about connecting elements as the glue that holds the different pieces of architecture together.

6.5 Data

Data is essential for all computing. In the broad concept data describes the instructions to the inputs and outputs from the system together with calculated dynamic runtime compounds that live shortly in the stack. In this chapter different data types are studied on classification

Fielding (Fielding, 2000: 11) defines data in context of architecture “A datum is an element of information that is transferred from a component, or received by a component, via a connector”. That is in line with the concept that components are their interfaces when looked outside. Internal data structures and databases are hidden from the scope.

In context of microservices simple in-wire-protocols are used together with self-describing data. Basic model with networking software is message based communication. These messages are also known as Protocol Data Unit (PDU). PDUs are often events that require some action from the component. Depending of the context and from the API. There are

two type of messages those whom change the state of the service by setting some values or requesting action that will perform the change and query messages don't change the state. Query messages are respond from the current state of the service. Simple example is ping-pong request-response pair of PDUs.

There are also some situation where an entire component or service is packed into a data-element entity as for example fetching or setting latest or certain version of binary or library from a service to be used or deployed locally.

6.6 Configuration

Fielding (Fielding, 2000: 12) summarized "A configuration is the structure of architectural relationships among components, connectors, and data during a period of system run-time". Microservice architecture aims for flexible configurations. Lewis and Fowler (Lewis and Fowler, 2014) used the term Evolutionary Design in their article about microservices. Microservice architecture embraces the change trough the dynamic modular configurability.

Services are built in a way that they can be easily utilized from existing applications with standard web connectors. The evolution starts often from small. There are many reasons why to integrate an external service to provide some feature. Some platforms even have plugin model for that.

Heraclitus of Ephesus says, "you know, that all things move and nothing remains still" in a dialogue by Plato (Plato, 1921). Same notion could one make about the networking software systems. The configuration changes when service modules are split or merged together in order to isolate the changes. Splitting can happen for example when some part of the module needs to scale by replication. Merging is an option if to modules often changed together.

Services can be updated or change to ones with equivalent API. Deployment of updated service should be done without ramping down the existing ones. Clients can be divided into groups that some are served with the new implementation sooner when requesting for service location.

Traditional solution to manage configurations is to use versioning with APIs (Lewis and Fowler, 2014)

6.7 Testing and monitoring

Testing of microservices working together is hard in conventional ways. Unit Tests could be done to check the basic functionality from regression in service level. There can be a test environment for the application level testing with the services and applications running tests cases to simulate the system level interaction.

By design a service call could fail at any time complicating the application logic as it needs to handle these gracefully. One method to assert this issue is to implement more services. Resource locator service for example can be used in situation that service is unavailable or slow to locate working service. Resource locator can monitor the status and handle load balancing while offering path to alternative instances.

Watchdog type service can be used monitor service instances and restart them if they crash. Also performance and load monitor implementation can potentially signal deployment service to scale by launching more certain service instances that would register themselves to resource locator service and watchdogs during their initialisation phase.

Logging and telemetry are essential on monitoring end debugging networks software. Bootstrapping your implementations different phases with telemetry and logging. Getting insight of operational phase, start-up, initialisation, normal processing, re-initialisation and shutdown is curtail on operating the system.

If one does not want to go through the hassle of implementing own telemetry solutions there are many good cloud base solutions available for example New Relic[27] one to mention. They can be integrated with API and Licence key and endpoint.

When operating in cloud environment logging could be left on also in production due cheap storage available. Logging reduce debugging time when problems occur. When working with multi server infrastructure establishing remote connection to the troubled server instance in order to check for logs can be time consuming. Implementing centralized logging can give advantages on monitoring them and also automating actions based on log entries. When wanting to take actions based on log entry stream it is important to differentiate informative log entries and action log entries in addition to normal log levels. (Dykstra, Anderson and Wasson, 2014)

Some companies that operate with cloud service have taken the resiliency readiness even further by developing tools like Netflix's Chaos Monkey[28] that would create random instance failures.

[27] http://newrelic.com/
[28] https://github.com/Netflix/SimianArmy

7. POWERSHELL BASED IMPLEMENTATION

In this chapter the ad hoc implementation of the scene verification service is documented. This project is called PSRobotVision. The solution utilizes command-line tooling wrapped into self-exposing http-microservice. One of the guidelines was that no external software was to be installed on to the host. The stand-alone self-contained target was bend by utilizing preinstalled PowerShell executable as host and gate way to run managed code in the system.

7.1 Introduction to PowerShell

PowerShell (PS) is a powerful tool with dynamically typed object-oriented scripting language that is capable of creating and initiating .NET classes and Windows Management Instrumentation (WMI)/COM objects. It can be used to create WPF graphical user interfaces and handle events. There is support for delegates and DLL loading. It also offers library for other processes to create and execute its workspaces.

In CLI usage it is common to use the Commandlets (Cmdlet) which are functions with cmdlet binding tag forcing the implementation to certain guideline. Cmdlets can also take parameters from the pipeline. Function can use positional and named parameters:

```
> (Get-Process "powershell")[0].Name
Powershell
> (Get-Process -Id $pid)[0].Name
Powershell
```

The syntax and return value (pipeline) handling can take some time to get used to. PowerShell engine also tries its best to convert objects and values implicitly. PowerShell itself is interesting and would probably deserve a paper of its own.

7.2 Start-up

As a side project for PSRobotVision a executable that save its own filename and swaps the extension to “ps1” was made starting powershell process with the script as a parameter and passes its own arguments to that as well. The starter process also sets the powershell not to exit after execution, so the shell stays open and reduces frustration that might come when invoking batch or script files that just exits without time to read the output.

In Program 7 is a little trick to inline import all the scripts from the main script directory and its subdirectories. It is useful if other script files are not executing anything serious, just declarations. This allows project folder to have arbitrary structure.

```
$ScriptDir = Split-Path $MyInvocation.MyCommand.Path -Parent
# Load all ps1 scripts from the path
$CoreScripts = Get-ChildItem "$ScriptDir"  -filter "*.ps1" -Recurse
foreach($script in $CoreScripts)
{
    if($script.FullName -ne $MyInvocation.MyCommand.Path)
    {
        Write-Host "Importing " $script.FullName
        .$script.FullName
    }
    else
    {
        Write-Host "Skipped invocation script" $script.FullName
    }
}
```

Program 7. *Trick to include everything from project.*

7.3 Http-listener

For the PowerShell process to reference .NET libraries Add-Type cmdlet is used. Basic libraries are in the path and can be referenced by assembly name:

```
Add-Type -AssemblyName System.Web
Add-Type -AssemblyName System.Net.Http
```

An example of adding other managed DLL that is located in the project directory is demonstrated in snippet below this subsection with the log4net-logging library. It might seem that there is also some unnecessary code to find the location of the library to keep the level of freedom. A wrapper for powershell exposing logging cmdlets that utilize pipeline was made and it is located in a folder with the DLL and can be dropped to any project bootstrapped with the trick:

```
$searchDir = Split-Path $MyInvocation.MyCommand.Path -Parent;
$Log4netDll = Get-ChildItem "$searchDir" –filter "log4net.dll" -Recurse | Se-
lect-Object -First 1
Add-Type -Path $Log4netDll.FullName -Verbose
```

Now that the types are added one can initialize the HttpListener object. Opening a socket needs elevated rights for the shell. Script for checking for the administrative rights to be placed right at the start of execution is shown in Program 8. If it detects that there are no administrative rights for the user session it starts new PowerShell-process with same arguments that itself had with elevated rights and exists. For user point of view this might require a dialog to be pressed to grant the rights for the process to make changes on the computer.

```
if (!([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIden-
tity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Admin-
istrator))
{
    # We do not have administarive rights in this session
    # Start new powershell with admin rights same params
    $arguments = "-noexit -ExecutionPolicy unrestricted & '"
                              + $myinvocation.mycommand.definition + "'"
    Start-Process powershell -Verb runAs -ArgumentList $arguments;
    Stop-Process $pid;
}
```

Program 8. *Trick for elevating PowerShell session.*

In the Program 9 is an example how to create new objects with full name to New-Object cmdlet. Calling a method on object is done with the dot notation and brackets. Static members of class can be used with namespace in square brackets and double colons.

```
$Global:listener = $null;
$listener = New-Object System.Net.HttpListener;
$listener.Prefixes.Add('http://+:8000/');
$listener.AuthenticationSchemes =
                              [System.Net.AuthenticationSchemes]::Basic;
$Global:listener.Start();
```

Program 9. *Initializing and opening a HTTP socket with listener.*

The .NET System.Net.HttpListener object has GetContext() method that blocks until a new request is being received. In Program 10 the main service loop is shown without the break and restart logic. The loop is actually “loading” the implementation of Handle-Route –function from dynamicroute.ps1 for every request. Some test were done and it actually is not doing much extra if there is no change in the file. In development phase this allows rapid changes to be conducted to the logic with code-editor of choise. In “production” that can be commented out. The context variable holds System.Net. HttpListenerContext object witch has request and response objects that are used to serve the request.

```
while($true)
{
      # blocks until request is received
      $context = $Global:listener.GetContext();
      ."$ScriptDir\logic\dynamicroutes.ps1"
      Handle-Route ($context);
```

Program 10. *Setting up the Handling for HTTP requests.*

In the request handling function some basic authentication is done by reading the Authorization header. Helper function for parsing the possible query parameter value. Composing custom PSObject for result and converting it to JSON or serving “static” files by reading them into memory stream and writing the byte stream to response buffer.

7.4 Camera

In order for PSRobotVision to have vision it needed eyes. A Microsoft® LifeCam Studio™ USB-camera with high definition was connected as a hardware component.

Some experiments of using Aforge libraries from PowerShell was made, but there were stability issue and not too much time for debugging. More about utilizing Aforge on .NET on chapter 8

As time was essence a search for light weight DirectShow engine as in form of standalone executable was conducted and RobotEyez.exe[29] (44KB) was found. Ted Burke has written this with C-code to be used by his students while they learn how to manipulate gray-scale bitmaps and control robots.

RobotEyez accepts parameters for more advanced usage and had name spot on for this project as well. In Program 11 is the actual line of code that takes a color photo with requested connected camera with dimension.

```
RobotEyez /devnum $devnum /width $width /height $height /bmp
```

Program 11. *Image capturing*

The frame is then read into memory, converted to png-format and saved to disk with given resource name. Take-Photo function uses GetImagePath-helper function to combine device number and resource name. The path to new image is returned to be used by the invoker.

```
$imgpath = "$ScriptDir\frame.bmp"
$destf = GetImagePath $devnum $filename "png";

$frame = [System.Drawing.Image]::FromFile($imgpath, $true);
if($frame)
{
   $frame.save($destf, [System.Drawing.Imaging.ImageFormat]::Png )
   $frame.Dispose();
}
Remove-Item -Force frame.bmp;
return $destf;
```

Program 12. *End of Take-Photo function.*

[29] https://batchloaf.wordpress.com/2011/11/27/ultra-simple-machine-vision-in-c-with-roboteyes/

7.5 Comparison and crop

When it comes to command-line image manipulation there is ImageMagick[30] that has cross-platform solution and good performance.

ImageMagick is offering portable stand-alone binary archives to download. It seems they have gotten new boost in recent years as digital photography has become mainstream. Some years ago there was a bit of hassle to install and take the binaries from the installation folder in order to use them. CodePlex[31] has also .NET library in alpha phase.

To utilize the ImageMagic executables the folder needs to be added to environment path this is done in Program 13 through variable called env. Modifying this is safe as it is a copy for the process.

```
$env:path+=";$exePath\"
```

Program 13. *Appending environment variable to path.*

Comparison is done using compare.exe and invoking it through command prompt process, even though PowerShell is also capable of executing the command. With compare we use absolute error count as metric this is outputted to stderr (-metric AE)[32]. To soften the comparison –fuzz[33] parameter is given it uses percentage distance of colors to considering them equal. Two input file paths and one output file path is also given as positional parameter. Output file as seen in Figure 9 is gray picture with red pixels on top of the ones that are considered different. The camera was aiming at static object and was move by hand between the initial photo taken and the comparison. There are 2690 pixel difference with 30% fuzz. Request path was: /api/camera/1/compare/test1?d=620x240+0+0

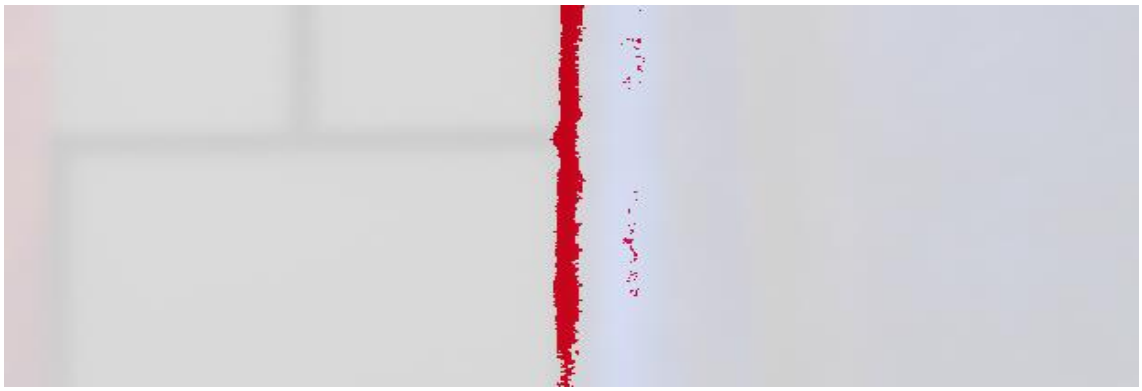

Figure 7. *Example of difference in the output file.*

[30] http://www.imagemagick.org/script/index.php
[31] http://magick.codeplex.com/
[32] http://www.imagemagick.org/script/command-line-options.php#metric
[33] http://www.imagemagick.org/script/command-line-options.php#fuzz

Cropping is done in similar fashion as comparison with imgconvert.exe original name was convert.exe, but this did clash with some command already in the path. Converter takes input file, dimensions and output file. Dimensions are in form of WIDTHxHEIGHT{+-}[X]{+-}Y so that aspect ratio is preserved X and Y are offsets (default +0+0). These can be given to the service as query parameters. A note that plus-signs are converted to spaces in the URL, so spaces are converted back to plus-signs in the method implementation if query string has them.

In Figure 10 we have an overview of how these different command-line tools are working together.

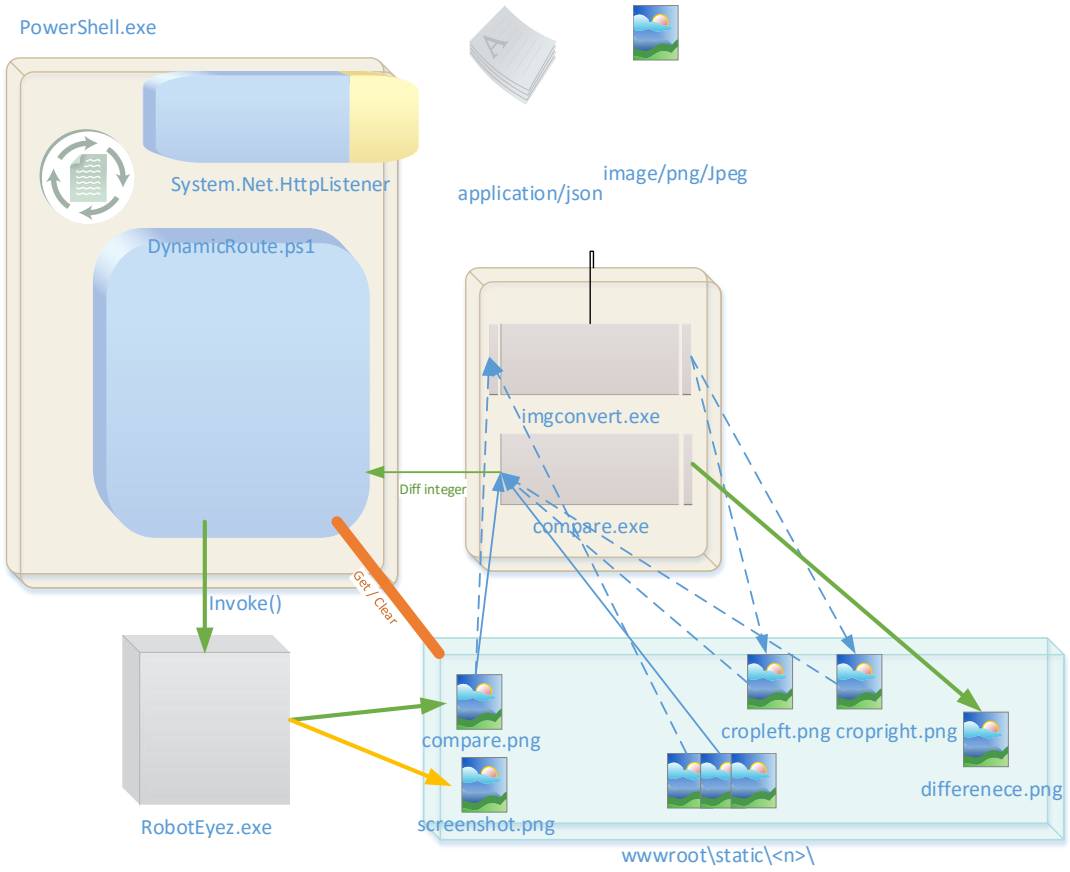

Figure 8. *Overview of the PowerShell solution*

8. ASP .NET 5 BASED IMPLEMENTATION

This chapter is about implementing the same functionality using the new .NET Core paradigm and ASP.NET 5. The name of this project is CameraService in contrast to PSRobotVision. There is slight additions to the API which is listed at 8.4. This project is also in GitHub repository[34].

8.1 Introduction to ASP .NET 5

ASP .NET name comes from the history of Active Server Pages and .NET refers to dot net ecosystem. Traditionally ASP NET projects have been hosted on IIS and it can be to heavy infrastructure for smaller services. ASP .NET 5 is one of the main product that benefits from the .NET Core project as it allows self-containing services to be deployed.

The development for this service is done mainly in Visual Studio 2015. Also some in place editing was done between the deployments in lab utilizing NotePad++. It was surprise to make change to C# .cs –file and restart the server and it was running

8.2 Start-up

When the DNX is launched with the command below. It gets the project directory and some arguments to pass to the host application as parameter. They are located in the same folder structure web.cmd:

```
exec "$DIR/runtimes/dnx-clr-win-x86.1.0.0-beta8/bin/dnx" --project
                  "$DIR/src/CameraService" --configuration Release web "$@"
```

When DNX starts it boots CLR loads the core libraries from its path and sets up AppDomain. DNX have resolved from the project.json in Program 14 command section command “web” Validates this Microsoft.AspNet.Server.Kestrel assembly with CLR. Then it then passes the execution to CLR by calling the managed entry point similar as 4.2.1 startup. CLR loads some of the managed assemblies in the bin-path and creates the IApplicationEnvironment-interface and calls the Kestrel functioning as application host entry point of and passes the execution on.

[34] https://github.com/ulkuniem/CameraService-aforge-dnx

```
{

  "webroot": "../../../wwwroot",
  "version": "1.0.0-*",
  "dependencies": {
    "AForge": "2.2.5",
    "AForge.Imaging": "2.2.5",
    "AForge.Video": "2.2.5",
    "AForge.Video.DirectShow": "2.2.5",
    "Microsoft.AspNet.Mvc": "6.0.0-beta8",
    "Microsoft.AspNet.Server.Kestrel": "1.0.0-beta8",
    "Microsoft.AspNet.StaticFiles": "1.0.0-beta8",
    "Microsoft.Framework.Logging": "1.0.0-beta8",
    "Microsoft.Framework.Logging.Console": "1.0.0-beta8",
    "Microsoft.Framework.Logging.Debug": "1.0.0-beta8"
  },
  "commands": {
    "web": "Microsoft.AspNet.Server.Kestrel --server.urls
                              http://localhost:5000;https://localhost:5001"
  },
  "frameworks": {
    "dnx451": {
      "dependencies": {},
      "frameworkAssemblies": {
        "Microsoft.VisualC": "10.0.0.0",
        "System.Data": "4.0.0.0",
        "System.Drawing": "4.0.0.0"
      }
    }
  },
}
```

Program 14. *Project.json*

8.3 Kestrel

This project is using Microsoft.AspNet.Server.Kestrel as Application host that is loaded first into DNX CLT. One of the reasons choosing Kestrel as a host is that its networking stack is implemented asynchronously with libuv[35][36].

Then Kestrel locates the CameraServices Startup.cs and by using System.Runtime.CompilerServices it gets compiled at runtime.

On the initialization phase of the Startup object constructor and configure methods are called. These calls have interfaces as reference and by calling those, extra services can be injected to be hosted. In the case of Asp.Net based hosts they derive from Microsoft.AspNet.Host and have the basic web service interface. Adding ASP.net MVC to host the CameraController.cs logic.

---

[35] https://github.com/aspnet/KestrelHttpServer/blob/dev/src/Microsoft.AspNet.Server.Kestrel/Networking/Libuv.cs

[36] https://github.com/libuv/libuv

```
public void ConfigureServices(IServiceCollection services)
{
      services.AddMvc();
}
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
{
      String staticFilePath = env.WebRootPath + "\\static";
      app.UseStaticFiles(new StaticFileOptions()
      {
FileProvider = new PhysicalFileProvider(staticFilePath),
                RequestPath = new PathString("/static")
     });
    app.UseDirectoryBrowser();
    // Add MVC to the request pipeline.
    app.UseMvc();
   // Init the camera static class
   Camera.AForgeStillWrapper VC = new Camera.AForgeStillWrapper(
env.WebRootPath);
}
```

Program 15. *Startup.cs from relevant parts.*

There is quite nice feeling when one gets a lot of results on few lines of code. The ASP.Net environment has the build in Kestrels implementation of directory browser can be handy with development and maintenance.

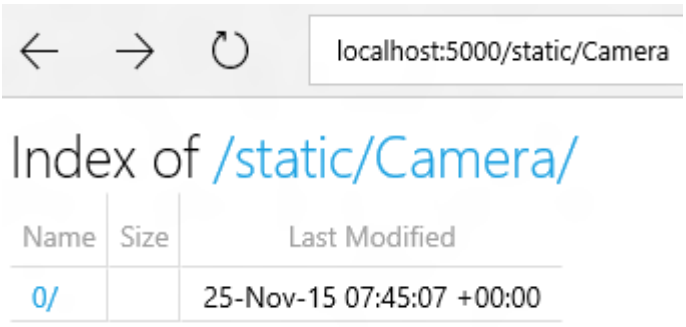

Figure 9. *Directory Browser*

8.4 HTTP API

The API of the CameraService exposing the basic functionality is listed in Program 16 .NET API specific are marked with *. If no reference picture compare results is 0 and the picture is set as new reference.

```
/api/camera/
/api/camera/0/compare/test1?th=50*
/api/camera/0/compare/test1
/api/camera/0/compare/test1?d=320x240+0+0
/api/camera/0/get/left/test1
/api/camera/0/get/right/test1
/api/camera/0/get/diff/test1
/api/camera/0/lastToNewReference/test1
/api/camera/0
/api/camera/0/clear/test1
*/api/camera/0/GetCameraControlProperties
*/api/camera/0/SetCameraControlProperties?Zoom=0&Exposure=auto&Focus=30
*/static/Camera/0/diff/test1.png
```

Program 16. *HTTP Endpoints*

IEnumerable objects through the http-api was too straight forward so a TemplateMessage-class was reused from older project that complies to Meteror Frameworks Distributed Data Protocol (DPP)[37]. In Program 17 a root of the API is handled /api/camera/ resulting list of video capturing devices and the resolution profiles and effective settings.

```
[HttpGet]
public ActionResult GetDevices()
{

       CameraService.DPP.TemplateMessage result =
                                          new DPP.TemplateMessage("result");
       result.id = CameraService.DPP.TemplateMessage.NextId();
      result.result = Camera.AForgeStillWrapper.ListDevices();
      return Content(result.stringify(), "application/json");
}
```

Program 17. *Examples of API implementation on ASP.NET MVC.*

In the Program 18 is example of API endpoint that takes a lot of query parameters. Only the id is bind to the API –end point and the other parameters are then used implicitly as query parameters. It is worth to mention that int is not nullable as it is a value type for this reason and also to allow *auto* as a parameter they are received as strings. Similar calls to SetCameraControlManual and SetCameraControlAuto is called for other parameter. The return value is list of changed properties with new values. If the property has enum none it is not set only manual/auto is touched. This is done in the AForgeCameraWrapper-class.

[37] https://github.com/meteor/meteor/blob/devel/packages/ddp/DDP.md

```
[HttpGet("{id}/SetCameraControlProperties")]
public ActionResult SetCameraControlProperties(int id, String Pan, String Tilt,
String Roll, String Zoom, String Exposure, String Iris, String Focus)
{
  List<Camera.AForgeStillWrapper.ContainerForCameraControlPropery> output = ..

  int pan = 0;
  if(!String.IsNullOrEmpty(Pan) && Int32.TryParse(Pan, out pan))
  {
      // Got Pan
      output.Add(Camera.AForgeStillWrapper.SetCameraControlPropertyManual(id,
                  AForge.Video.DirectShow.CameraControlProperty.Pan, pan));
  }
  else if(!String.IsNullOrEmpty(Pan) && Pan.ToLower().Equals("auto"))
  {
  output.Add(Camera.AForgeStillWrapper.SetCameraControlPropertyAuto(id,
                        AForge.Video.DirectShow.CameraControlProperty.Pan));
  }
  ..
}
```

Program 18. *Parsing query parameters for ASP.NET MVC.*

8.5 Camera

For the camera imaging Aforge[38] .NET Libraries were used. They are available in NuGet as packets and thus fitted to the architecture.

A directory Camera was created together with the AForgeStillWrapper.cs resulting in class called CameraService.Camera.AForgeStillWrapper.

The initial version was quite straight forward to create listing available video input devices and initializing video capture device.

8.5.1 Memory usage

The first implementation held bitmaps in a static array of the length of cameras attached. Cloning new frames to their places when they arrived. The idea was to have the latest frame at hand to be returned with GetFrame-method.

It basically works, but the amount of memory reserved started to kick in and GC was doing cleaning in periodic interval. Then next implementation asserted the problem by temporally referencing to old frame and cloning the new one in and disposing the old one got the memory allocation to stay steady in reasonable size. In aftermath could have used another temporal variable to hold the new one as it was being cloned. It might also bring other issues with saving bitmap that is being disposed in the other thread as was found out accidentally.

[38] http://www.aforgenet.com/

8.5.2 Multithreading

We also noticed some tearing in some pictures and this could be due to the fact that the camera is attached to a moving object (which it was) or the same block being reused for saving new frame. The frame rate with the test equipment was 30.

AForge is actually opening another thread for the capturing device to form the frames and call the delegate is shown:

```
SelectedDevice.NewFrame += delegate (object sender,
                                                NewFrameEventArgs eventArgs)
{
     FrameQueueArray[device].Enqueue((Bitmap)eventArgs.Frame.Clone());
};
```

Finally to assert the concurrency issues a custom class implementing *System.Collections.Concurrent.ConcurrentQueue* was created. As Appendix A shows a custom Enqueue and TryPeek method were created to assure that there is latest frame safely available and it gets cloned safely when requested. TryDeque is used when restart has occurred to clear the queue. One was set initially to be the default max size for this queue and it has worked well.

8.5.3 Auto shut down of unused equipment

At one point developing the feature to turn off unused cameras by using the delegate to check for the conditions. Not by surprise signaling for stop from that a thread does not really do much. A timer was then implemented to do this logic.

This prevent some cameras from heating as the service is kept always on. Timers callback functions implementation is in Appendix B. The timer is set so that it will reinitiate itself after trigger as there are no strict timing involved no correction to the interval is made of the actual execution of the callback function itself. There are WaitForStop() call that may possibly block until the background thread is at state to check for the stop signal.

8.6 Comparison and crop

Comparison that is being done in this service with AForge.Imaging.Filters.ThresholdedDifference[39] filter. Threshold (th) being a the accepted difference between pixels in overlay and applied image. In contrast to the imagemagick this has differentiating pixels marked as white in a black background. Result is obtained from the filter.WhitePixelsCount variable:

```
filter = new AForge.Imaging.Filters.ThresholdedDifference(th);
filter.OverlayImage = left;
resultImage = filter.Apply(right);
..
result.result = filter.WhitePixelsCount;
```

Cropping is done as shown in Program 19 with System.Drawing class methods in memory drawing from the reference image to new bitmap.

```
System.Drawing.Rectangle cropArea = new Rectangle(x, y, widht, height);

// Crop left
Bitmap nleft = new Bitmap(cropArea.Width, cropArea.Height);
Graphics gr = Graphics.FromImage(nleft);
gr.DrawImage(left, -cropArea.X, -cropArea.Y);
gr.Dispose();
left.Dispose();
left = nleft;
```

Program 19. *Cropping in .NET. using System.Drawing .Graphics*

[39] http://www.aforgenet.com/framework/docs/html/322123cf-39df-0ae8-6434-29cceb6a54e1.htm

9. COMPARISON OF SOLUTION

In this chapter some measurements and tests are run against both implementations. In the Sections 9.1 and 9.2 physical characteristics of the code and runtime space is observed from filesystem. Section 9.3 is for functional testing and measurement of the behavior from clients side of view.

9.1 Amount of code

For the the quantitative code analysis a Program 20 was written in PowerShell, when it will promt for parameters at execution.

```
param(
    [Parameter(Mandatory=$True)]
    [string]$root,
    [Parameter(Mandatory=$True)]
    [string]$filter
)

$codeFiles = Get-ChildItem $root -filter $filter -Recurse $Lines = 0;
$Files = 0;
$Lines = 0;
$Words = 0;
$Chars = 0;
foreach($codeFile in $codeFiles)
{
    if($codeFile.Name -eq (Split-Path $MyInvocation.MyCommand.Path -Leaf ))
    {continue;}
    $Files += 1;
    $mes = $codeFile | Get-Content | Measure-Object -Line -Word –Character;
    $Lines += $mes.Lines;
    $Chars += $mes.Characters;
    $Words += $mes.Words;

    Write-Host "$($codeFile.Name) Lines: $($mes.Lines) Words: $($mes.Words)
Characters: $($mes.Characters)";
}
Write-Host "Total Files: $Files Lines: $Lines Words: $Words Chars: $Chars";
```

Program 20. *Code Analysis script*

When the Program 20 is run against PowerShell project directory with filter *.ps1 the output is in Program 21.

```
cmdlet count.ps1 at command pipeline position 1
Supply values for the following parameters:
root: C:\Temp\PSRESTRobotVision
filter: *.ps1
Server.ps1 Lines: 134 Words: 407 Characters: 4250
Log4Net.ps1 Lines: 151 Words: 289 Characters: 2940
Message.ps1 Lines: 469 Words: 2001 Characters: 15536
dynamicroutes.ps1 Lines: 883 Words: 2687 Characters: 65775
Vision.ps1 Lines: 142 Words: 438 Characters: 5766
Total Files: 5 Lines: 1779 Words: 5822 Chars: 94267
PS C:\Temp\PSRESTRobotVision>
```

Program 21. *PowerShell project code statistics.*

When the Program 20 is run against PowerShell project directory the output is in

```
cmdlet count.ps1 at command pipeline position 1
Supply values for the following parameters:
root: C:\Temp\CameraService\Release\approot\src\CameraService
filter: *.cs
Startup.cs Lines: 66 Words: 226 Characters: 2702
AForgeStillWrapper.cs Lines: 352 Words: 1063 Characters: 15610
CameraController.cs Lines: 449 Words: 1364 Characters: 20819
TemplateMessage.cs Lines: 309 Words: 2173 Characters: 14412
AssemblyInfo.cs Lines: 20 Words: 113 Characters: 953
Total Files: 5 Lines: 1196 Words: 4939 Chars: 54496
PS C:\Temp\PSRESTRobotVision>
```

Program 22. *.NET project code statistics.*

9.2 Memory footprint

At runtime idling the PowerShell.exe hosting the app is flat around lining around 28MB on the reference PC. It is not increased by API calls as most of the data is stored in disk and only read when needed. Dnx.exe is idling at 12 MB and when camera is active the allocation settles around 70MB. In lab environment where focus, exposure and zoom are set manually the camera thread can be stopped in under a secod.

9.3 Network throughput and latency

JMeter[40] is multipurpose open source tool for load testing functionality and measure performance. It has pretty simple interface to add threads and loops with requests. Default values for address and authentication can be set. It also supports proxies so more in depth analyze of under the application layer networking can be achieved. It support also response assertions so automation triggering on received content can be built.

[40] http://jmeter.apache.org/

Basic loops were created and run against both services operating in the same machine result are visualized in Figure 12 and 13. The API calls are a subset of Program 16. The PSRobotVision does not have implementation for static files, although it could be added to the dynamic routes as an endpoint. In the Figure 12 big variation is seen on the cases where RobotEyez is called. It seems that taking picture with camera is the common factor.

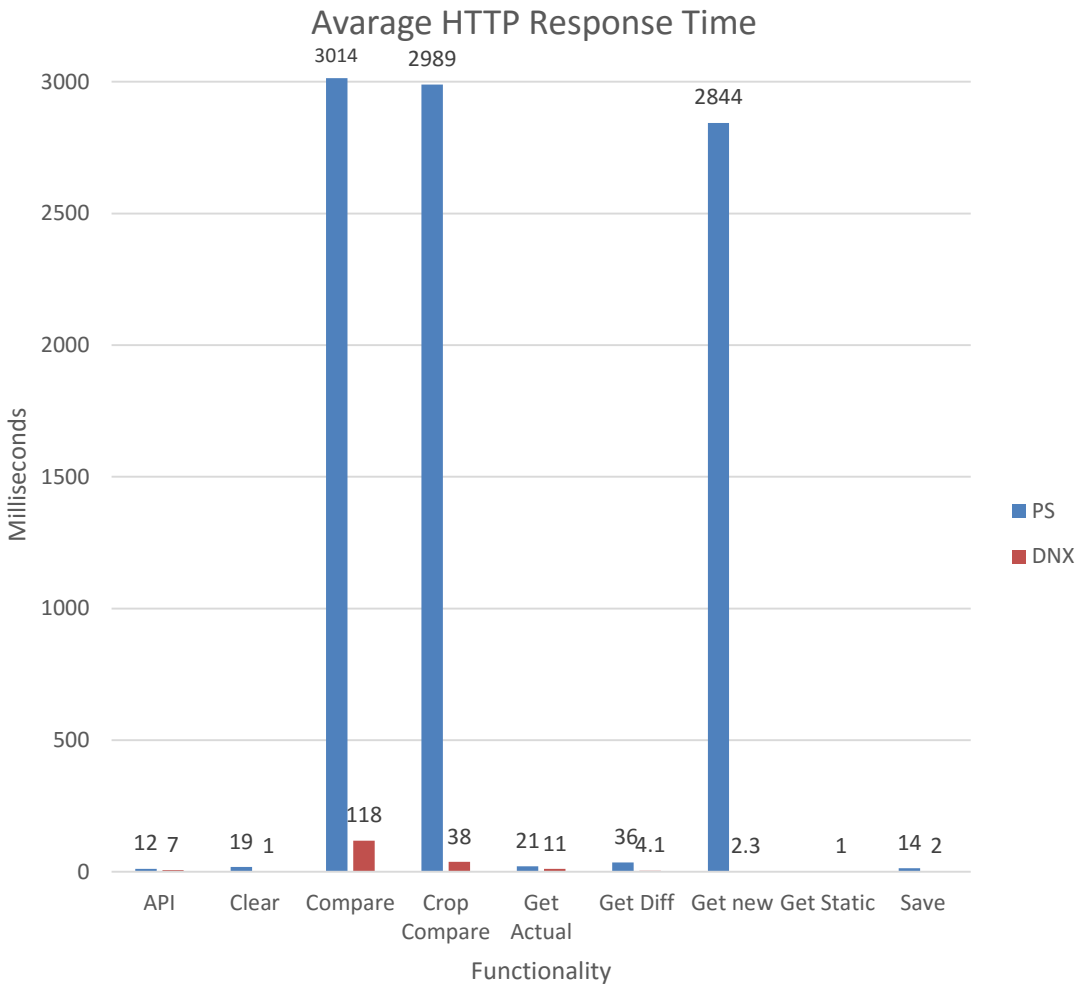

Figure 10. *Average HTTP Response Time Comparison*

It seems that taking picture with camera is the common factor. In Figure 13 the Get New value is reduced from other camera related cases with PS piles. .NET image capture does not take time hardly at all as it is already in the memory. There are conversion to Jpeg-format in all cases except the Get Static one. It seems that comparison takes most of the time and cropping the image to smaller is in both solutions beneficial. Although the latencies here are at acceptable range.

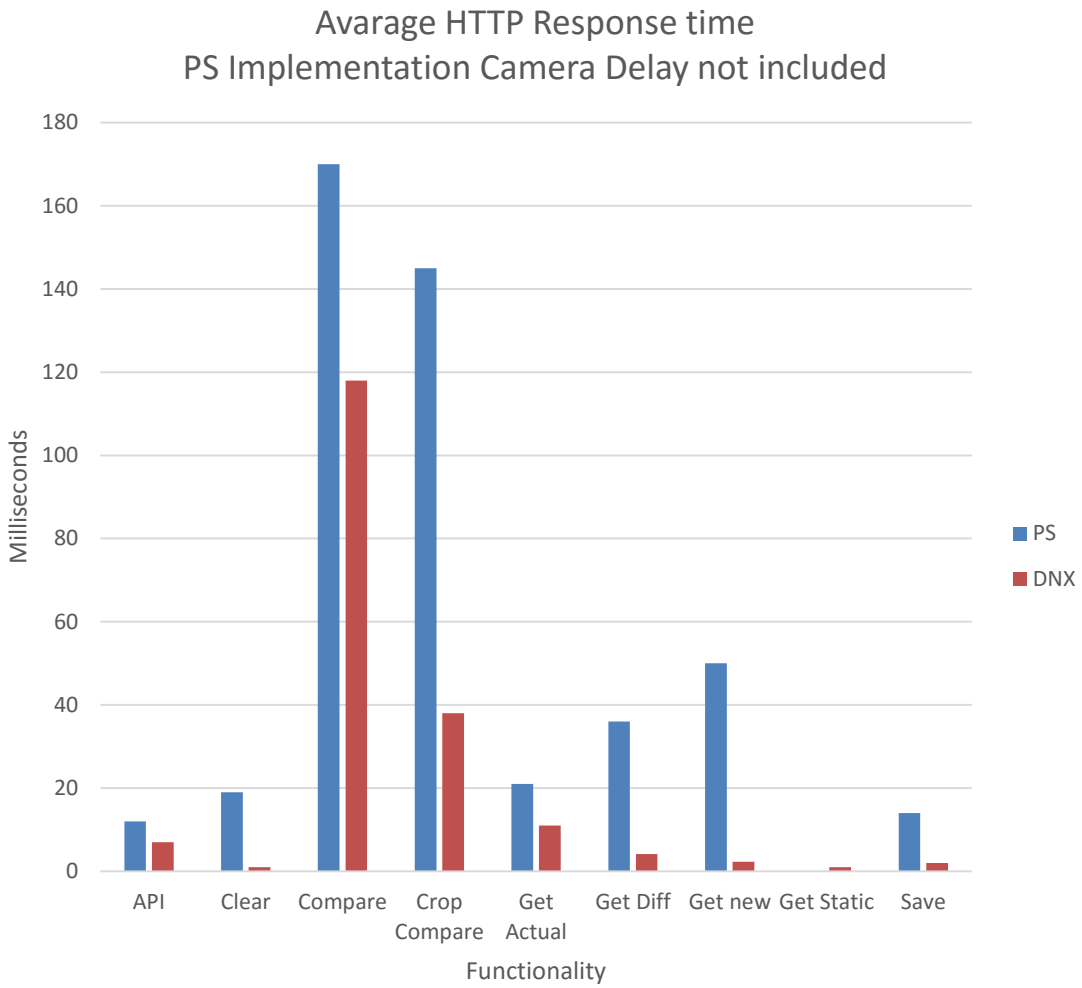

Figure 11. *Average HTTP Response Time Comparison without Camera Delay*

It seems that implementing the AForge camera to PSRobotVision would bring it to close range to a supposedly very fast production ready ASP.net runtime. Porting the AForgeCameraWrapper.cs to PowerShell .ps1 manually is one solution, but maybe more straight forward would be to compile it to managed DLL and load it to PowerShell session.

10. CONCLUSION

Both of the solutions works as expected the needed performance gain of using in-memory-solution for camera image management against an implementation of stand-alone command-line tooling that writes the relatively large bitmap to a file and performs the image processing on filesystem side was noticeable. Similarities in the development processes are found on the new .Net execution environment while the building phase is mixed into memory and runtime.

In future utilization of the new compilers that .NET Core offers to build even native classes for PowerShell to use. To start with the AForge camera wrapper from this thesis. The libuv-based Kestrel-server-application host performs well on I/O and an effort will be made to bootstrap that into PowerShell session to be utilized in smaller scale Microservice projects.

There is a lot of potential in many solutions for stand-alone self-contained software that is not IIS-size or needs to be distributed to many hosts. There will be a lot of potential in the ASP.NET 5 and .NETs self-containing execution environment also outside cloud computing.

While there were some dependency injection in the bootstrapping phases of .NET execution, an idea about service hosting the implementations of the method for the client. The client would get the implementation from the interface and bootstrap it in its own runtime and call the entry routine. There might be a place for such in test automation side as the actual test code should be kept fairly simple and updates to external interfaces are needed to coordinate to them anyway via repository dependencies.

Implementation injection from server side would be fairly simple to do in PowerShell as the client can just invoke web-request and download the file to where and under what name it pleases and the dot source the file to its own runtime. PowerShell might not even need an entry point as the script can have bare instructions. This could actually open a scenario where the server can actually dictate what the devices under test executes in centralized manner, as at the moment it is fully distributed.

With the gathered experiences using AForges imaging libraries experiments of more sophisticated machine vision and text recognition can be done by extending the HTTP API.

REFERENCES

ASP.Net Team (2014) *The Next Generation of.NET – ASP.NET vNext*, 12 May, [Online], Available: http://blogs.msdn.com/b/dotnet/archive/2014/05/12/the-next-generation-of-net-asp-net-vnext.aspx.

Basu, A., Buch, V., Voegels, W. and von Eicken, T. (1995) 'U-Net: A User-Level Network Interface for Parallel and Distributed Computing', Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain, Colorado.

Bhushan, A. (1971) *RFC 114. A FILE TRANSFER PROTOCOL*, MIT Project MAC, Available: http://www.ietf.org/rfc/rfc0114.txt.

Bosch, J. (2013) *Speed and Innovation (Seminar Presentation)*, [Online], Available: http://www.tut.fi/idcprod/groups/public/@l814/@web/@p/documents/liit/p044247.pdf.

Brander, S. (1996) *RFC 2026. The Internet Standards Process -- Revision 3*, Network Working Group, Available: http://www.ietf.org/rfc/rfc2026.txt.

Cheng, Y., Chu, J., Radhakrishnan, S. and Jain, A. (2014) *RFC 7413. TCP Fast Open*, IETF, Available: http://www.ietf.org/rfc/rfc7413.txt.

Cheshire, S. (1996) *It's the Latency, Stupid*, May, [Online], Available: http://rescomp.stanford.edu/~cheshire/rants/Latency.html [14 May 2015].

Crocker, S. (1969) *RFC 1. Host Software*, IETF, Available: http://www.ietf.org/rfc/rfc0001.txt.

Dykstra, T., Anderson, R. and Wasson, M. (2014) *Building Real-World Cloud Apps with Azure*, 12 June, [Online], Available: http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/.

ECMA International (2011) *ECMA-262. ECMAScript Language Specification*, June, [Online], Available: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

Fairhurst, G. (2008) *The User Datagram Protocol (UDP)*, 19 Nov, [Online], Available: http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html.

Fette, I. and Melnikov, A. (2011) *RFC 6455. The WebSocket Protocol*, IETF.

Fielding, R.T. (2000) *Architectural Styles and the Design of Network-based Software Architectures*, UCLA.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999) *RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1*, IETF, Available: http://www.ietf.org/rfc/rfc2616.txt.

Glass, R., (2003) *Facts and Fallacies of Software Engineering*, Addison-Wesley Professional.

Gravelle, R. *Comet Programming: the Hidden IFrame Technique*, [Online], Available: http://www.webreference.com/programming/javascript/rg30/index.html [17 May 2015].

Grigorik, I. (2013) *High Performance Browser Networking: What every web developer should know about networking and web performance*, O'Reilly Media, Inc.

Grossman, J. (2003) *Cross-Site Trancing (XST)*, 20 Jan, [Online], Available: http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf [24 Nov 2015].

Hartl, A. (2010) *C# Tips & Tricks: Immutable Types*, [Online], Available: http://tipsandtricks.runicsoft.com/CSharp/Immutables.html [21 Nov 2015].

Housley, R., Crocker, D. and Burger, E. (2011) *RFC 2026. Reducing the Standards Track to Two Maturity Levels*, IETF, Available: http://www.ietf.org/rfc/rfc6410.txt.

Huffstutter, P.J. (2000) *Dot-Com Parties Dry Up*, Los Anglese Times, Available: http://articles.latimes.com/2000/dec/25/news/mn-4559.

Kean, D. (2015).*NET Compiler Platform ("Roslyn") Overview*, [Online], Available: https://github.com/dotnet/roslyn/wiki/Roslyn%20Overview [21 Nov 2015].

Krill, P. (2008) *JavaScript creator ponders past, future*, 23 June, [Online], Available: http://www.infoworld.com/article/2653798/application-development/javascript-creator-ponders-past--future.html.

Kurose, J.F. and Ross, K.W. (2011) *Computer Networking a top-down approach*, 5th edition, Pearson.

Landwerth, I. (2014) *Introducing.NET Core*, 4 Dec, [Online], Available: http://blogs.msdn.com/b/dotnet/archive/2014/12/04/introducing-net-core.aspx [22 Nov 2015].

Lewis, J. and Fowler, M. (2014) *Microservices*, 25 March, [Online], Available: http://martinfowler.com/articles/microservices.html.

Marathe, N. (2015) *An Introduction to libuv*, 7 February, [Online], Available: http://nikhilm.github.io/uvbook/An%20Introduction%20to%20libuv.pdf.

McIlroy, M.D., Pinson, E.N. and Tague, B.A. (1978) 'Unix Time-Sharing System Forward', *The Bell System Technical Journal*, vol. 57, no. 6, July, pp. 1889-1904, Available: http://cobweb.cs.uga.edu/~eileen/1730/Readings/UNIX_Foreword.pdf.

Meier, J.D., Vasireddy, S., Babbar, A., Mariani, R. and Mackman, A. (2004) *Improving.NET Application Performance and Scalability*, May, [Online], Available: https://msdn.microsoft.com/en-us/library/ms998547.aspx#scalenetchapt05_topic9 [24 Nov 2015].

Melvin, J.T. and Watson, R., (1971) *RFC 97. A FIRST CUT AT A PROPOSED TELNET PROTOCOL*, SRI-ARC, Available: http://www.ietf.org/rfc/rfc0097.txt.

msdn (2012) *Common Language Specification*, [Online], Available: https://msdn.microsoft.com/library/12a7a7h3(v=vs.100).aspx [22 Nov 2015].

Myerson, T. (2015) *The next generation of Windows: Windows 10*, 21 January, [Online], Available: http://blogs.windows.com/bloggingwindows/2015/01/21/the-next-generation-of-windows-windows-10/ [14 May 2015].

O'Hanlon, C. (2006) 'A Conversation with Werner Vogels', *Queue - AI*, vol. 4, no. 4, May, pp. 14-22.

Pizzo, M., Handl, R. and Zurmuehl, M. (2014) *Standard. OData Version 4.0 Part 1: Protocol*, OASIS, Available: http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.pdf.

Plato (1921) *Plato in Twelve Volumes, Vol. 12 translated by Harold N. Fowler*, London: Harvard University Press, Available: http://www.perseus.tufts.edu/hopper/text?doc=Perseus%3Atext%3A1999.01.0172%3Atext%3DCrat.%3Asection%3D402a.

Postel, J.B. (1980) *RFC 768. User Datagram Protocol*, University of Southern California / Information Sciences Institute, Available: http://tools.ietf.org/rfc/rfc768.txt.

Radhakrishnan, S., Cheng, Y., Chu , J., Jain, A. and Raghavan, B. (2011) 'TCP Fast Open', in *Proceedings of the 7th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, ACM.

Roth, D. (2015) *DNX Overview*, [Online], Available: https://docs.asp.net/en/latest/dnx/overview.html [21 Nov 2015].

Russell, A. (2006) *Comet: Low Latency Data for the Browser*, Mar, [Online], Available: http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/.

Saxena, N. (2014) *Using Visual Studio to build Universal XAML Apps*, 14 Apr, [Online], Available: http://blogs.msdn.com/b/visualstudio/archive/2014/04/14/using-visual-studio-to-build-universal-xaml-apps.aspx [22 Nov 2015].

Shaw, M. and Paul, C. (1997) 'A Field Guide to Boxology: Preliminary Classification of Architectural Styles for', Proceedings of the 21st International Computer Software and Applications Conference, Washington, D.C, 6-13.

Stubailo, S. and Avital, O. (2014) *DDP Specification*, 9 October, [Online], Available: https://github.com/meteor/meteor/blob/devel/packages/ddp/DDP.md [14 May 2015].

The Open Group (1999) *Distributed Data Management (DDM) Architecture [Document Number: C913]*, The Open Group Technical Standard.

Vogels, W.H.P. (2003) *Scalable Cluster Technologies for Mission-Critical Enterprise Computing*, Vrije Universiteit.

Wilder, J. (2012) *Centralized Logging*, 3 Jan, [Online], Available: http://jasonwilder.com/blog/2012/01/03/centralized-logging/ [14 May 2015].

Wolf, A.L. and Perry, D.E. (1992) 'Foundations for the Study of Software Architecture', *ACM SIGSOFT SOFTWARE ENGINEERING NOTES*, vol. 17, no. 4, October, p. 40.

APPENDIX A: FIXED SIZE QUEUE FOR .NET

```
public class FixedSizedQueue<T>
{
      ConcurrentQueue<System.Drawing.Bitmap> q =
                              new ConcurrentQueue<System.Drawing.Bitmap>();
      public int Limit { get; set; }

      public void Enqueue(System.Drawing.Bitmap obj)
      {
            q.Enqueue(obj);
            lock (this)
            {
                  System.Drawing.Bitmap overflow;
                  while (q.Count > Limit && q.TryDequeue(out overflow))
                  {
                        overflow.Dispose();
                  }
            }
      }
      public bool TryDequeue(out System.Drawing.Bitmap obj)
      {
            return q.TryDequeue(out obj);
      }
      public bool TryPeek(out System.Drawing.Bitmap obj)
      {
            System.Drawing.Bitmap tmp;
            lock (this)
            {
                  bool ret = q.TryPeek(out tmp);
                  if(tmp != null)
                  {
                        obj = (System.Drawing.Bitmap)tmp.Clone();
                  }
                  else
                  {
                        obj = null;
                  }

                  return ret;
            }
      }
}
```

APPENDIX B DEVIVCE SHUT DOWN CHECK CALLBACK

```
private static void TimerCallback(Object state)
{
  int device = 0;
  foreach (FilterInfo fi in VideoFilterInfoCollection)
  {
     if ((DateTime.Now - CameraLastUsedTime[device]).TotalMilliseconds >
                                                  MAX_INACTIVE_IN_MILLISECONDS)
     {
           if ((device >= 0 && device < VideoCaptureDeviceArray.Length))
           {
                 SelectedDevice = VideoCaptureDeviceArray[device];
                 if(SelectedDevice != null && SelectedDevice.IsRunning)
                 {
                       SelectedDevice.SignalToStop();
                       SelectedDevice.WaitForStop();
                 }
           }
      }
      ++device;
   }
   // No ob check in private
   _watchTimer.Change(TIME_INTERVAL_IN_MILLISECONDS, Timeout.Infinite);
}
```