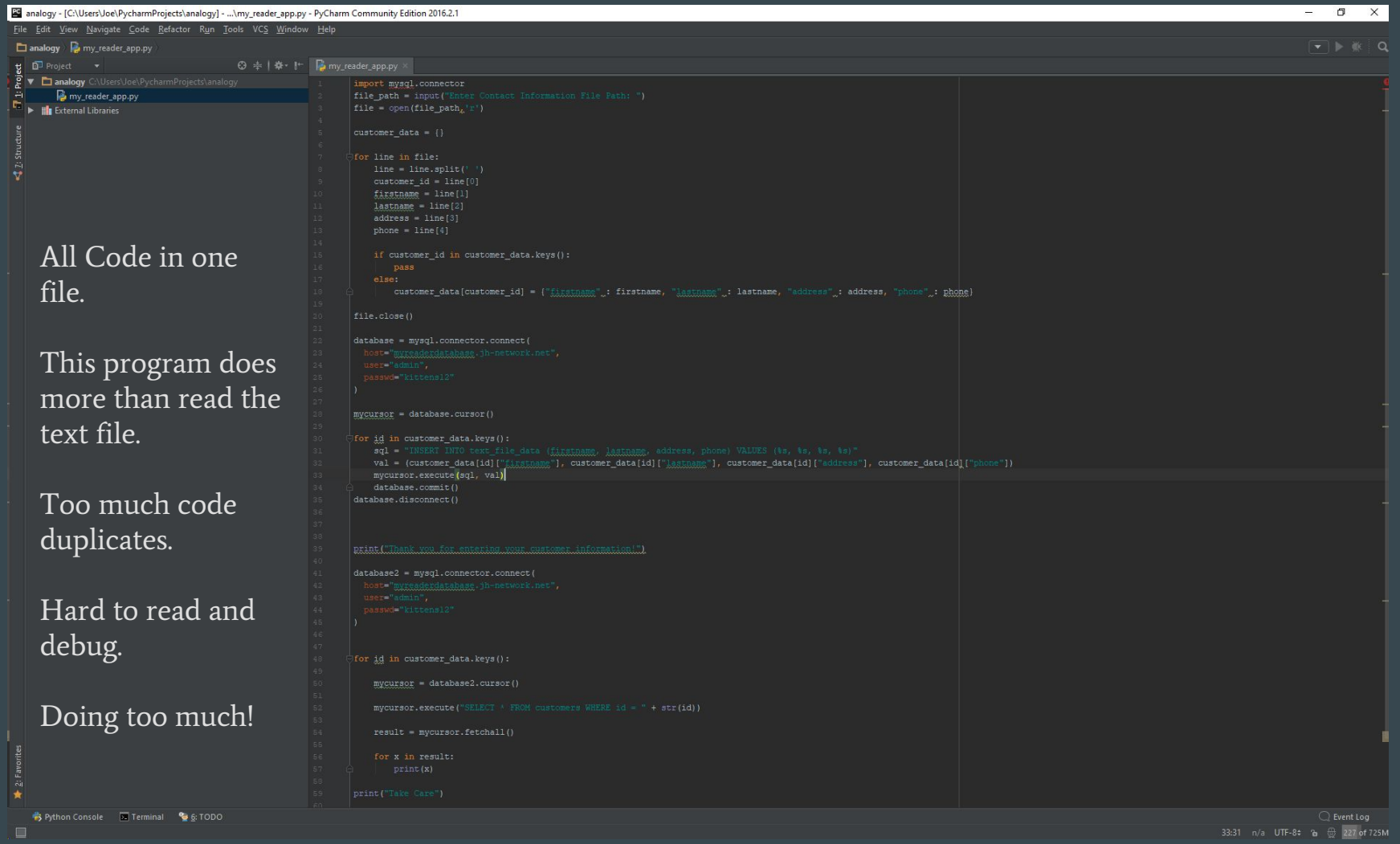
Monolithic Architecture vs Microservices Architecture ...

Joe Harris

Agenda

- Describe the Features, Pros, and Cons of both Monolithic Architecture and Microservices Architecture.
- Examples of both Architectures.
- Explain Containers vs Hypervisors.
- Explain how the Cloud and Cloud Computing fits into this.
- Questions and Answers.

Let’s Start with Something Seemingly Unrelated so I can Successfully Confuse you Immediately



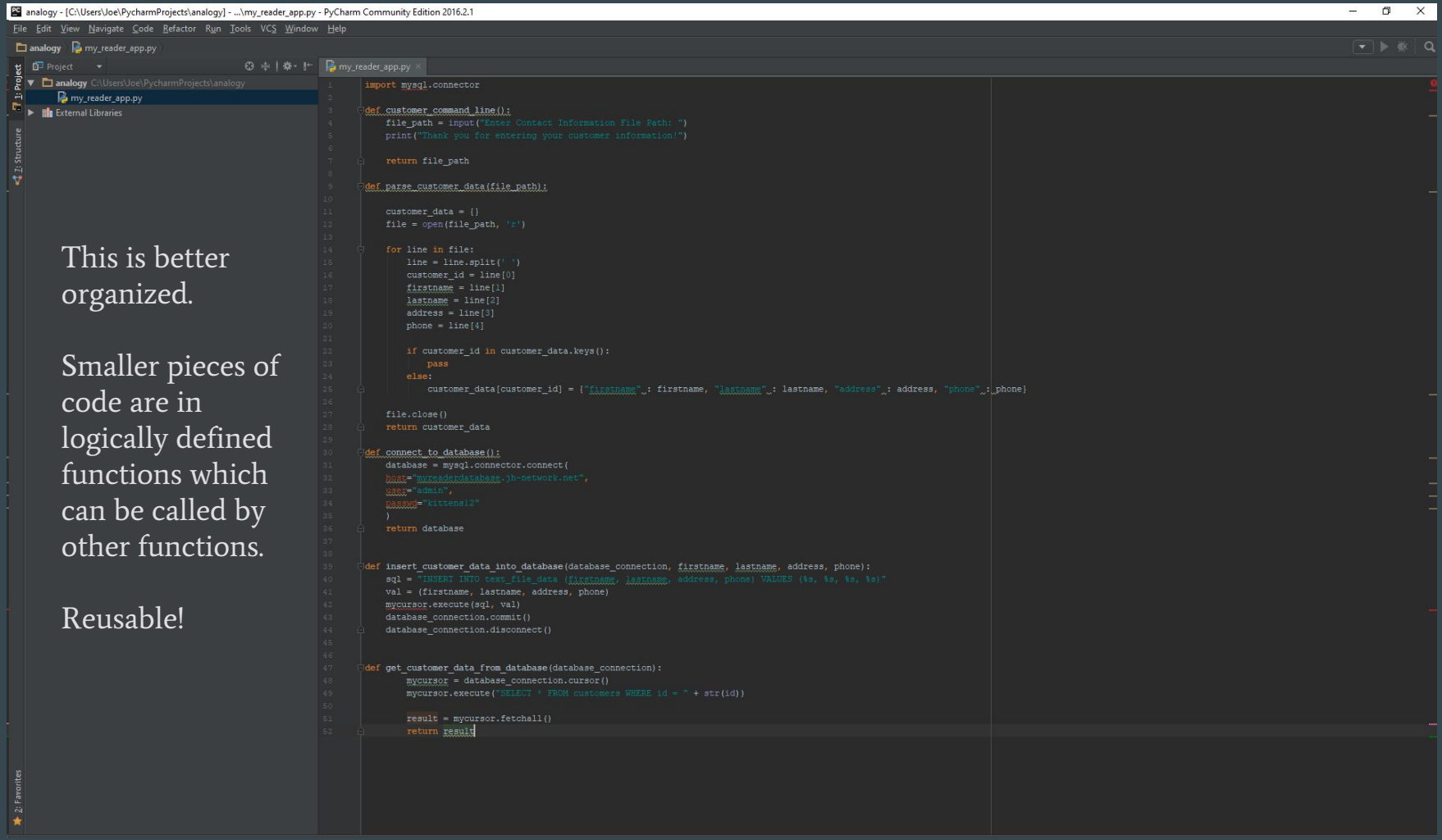
All Code in one file.

This program does more than read the text file.

Too much code duplicates.

Hard to read and debug.

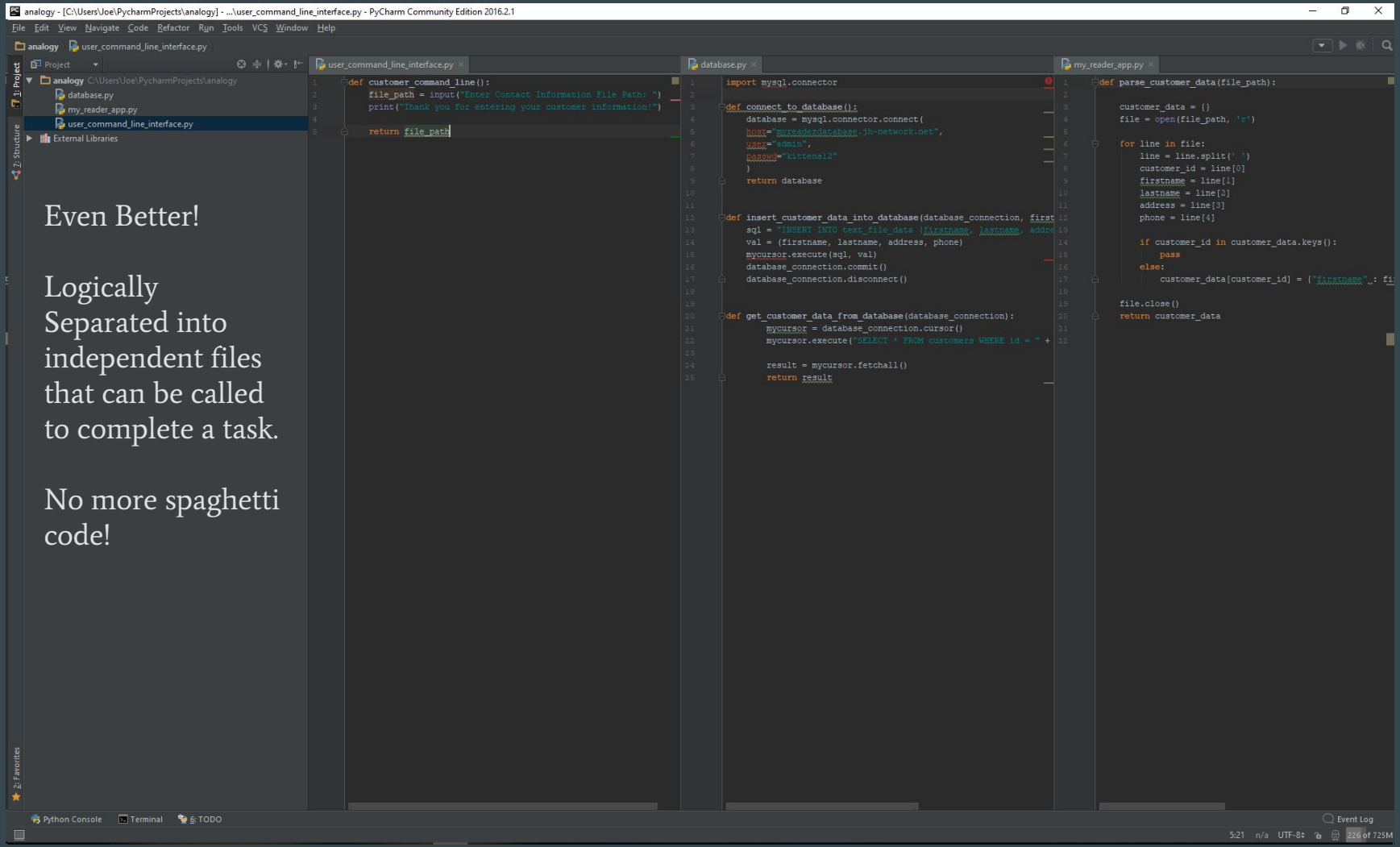
Doing too much!



This is better organized.

Smaller pieces of code are in logically defined functions which can be called by other functions.

Reusable!



Even Better!

Logically Separated into independent files that can be called to complete a task.

No more spaghetti code!

Think about this organizing approach to code while we talk about the different architectures

What is Monolithic Architecture?

Monolithic Architecture

An approach to writing software that is built as an interwoven, *interdependent*, single unified unit. Usually consists of three parts: a database, a client-side user interface (consisting of HTML pages and/or JavaScript running in a browser), and a server-side application. The server-side application will handle HTTP requests, execute domain-specific logic, retrieve and update data from the database, and populate the HTML views to be sent to the browser.

Single package that has all required components and services included.

What is Microservices Architecture?

Microservice Architecture

An approach to developing a single application as a suite of small *independent* services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services.

Independent pieces solely responsible for its own part of the greater application.

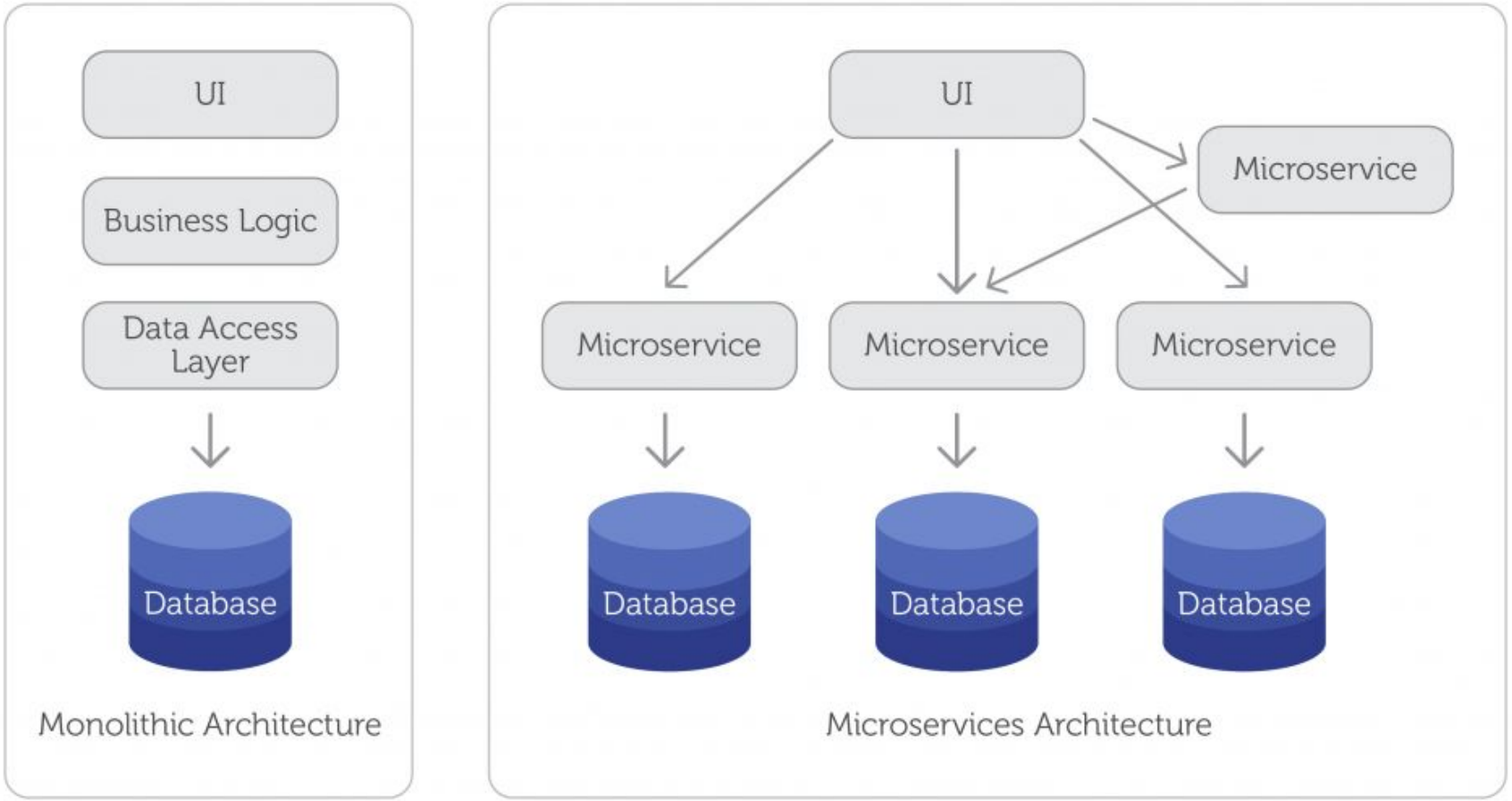
The idea of Microservices is not really a new one in Computer Science. Ken Thompson and Dennis Ritchie (*nerds*) outlined the idea of what we call Microservices in their *Unix Design Philosophy*.

In short, the Unix Design Philosophy can be summed up in one statement:

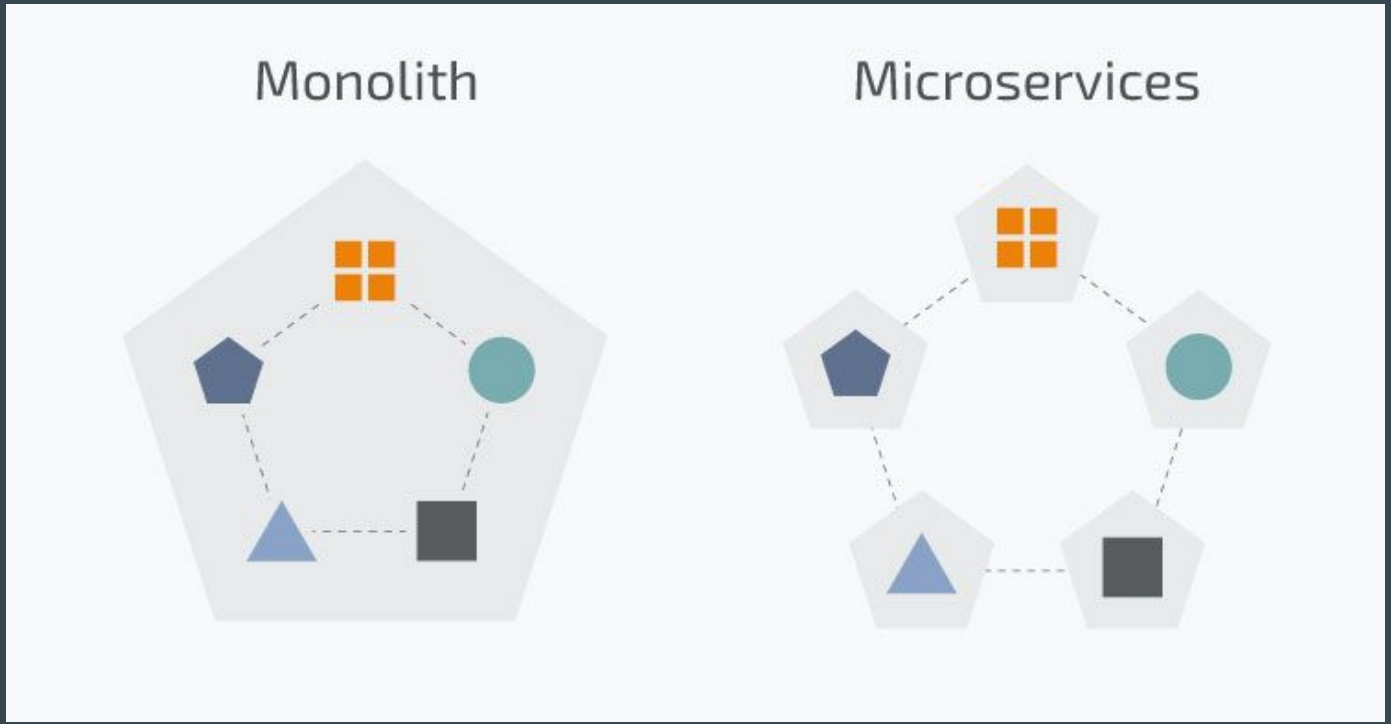
Do one thing well.

This is the idea of Modularity which refers to a system that is composed of components (modules) that can be fitted together or arranged in a variety of ways.

Sounds a lot like the idea behind Microservices... What is old becomes new again...


UI
Business Logic
Data Access Layer
Database
Monolithic Architecture
UI
Microservice
Microservice
Microservice
Microservice
Database
Database
Database
Microservices Architecture

Which Architecture is Better?



Key Features of a Monolith

- Code for the entire system is stored in one codebase. Although the code may be divided into classes, and packages and have good OOP structure, it is not split into modules for separate compilation. - *One Unit.*
- Since the code is packaged in a single version, to update a feature it is necessary to stop the whole system, roll out the new software and then restart it.
- Easy and reliable communication within the application via internal procedures.
- Every new release of a monolith means that it gets bigger in size. Becomes harder to manage.
- Scaling one piece of the application requires the entire application to scale since it is all *interdependent.*

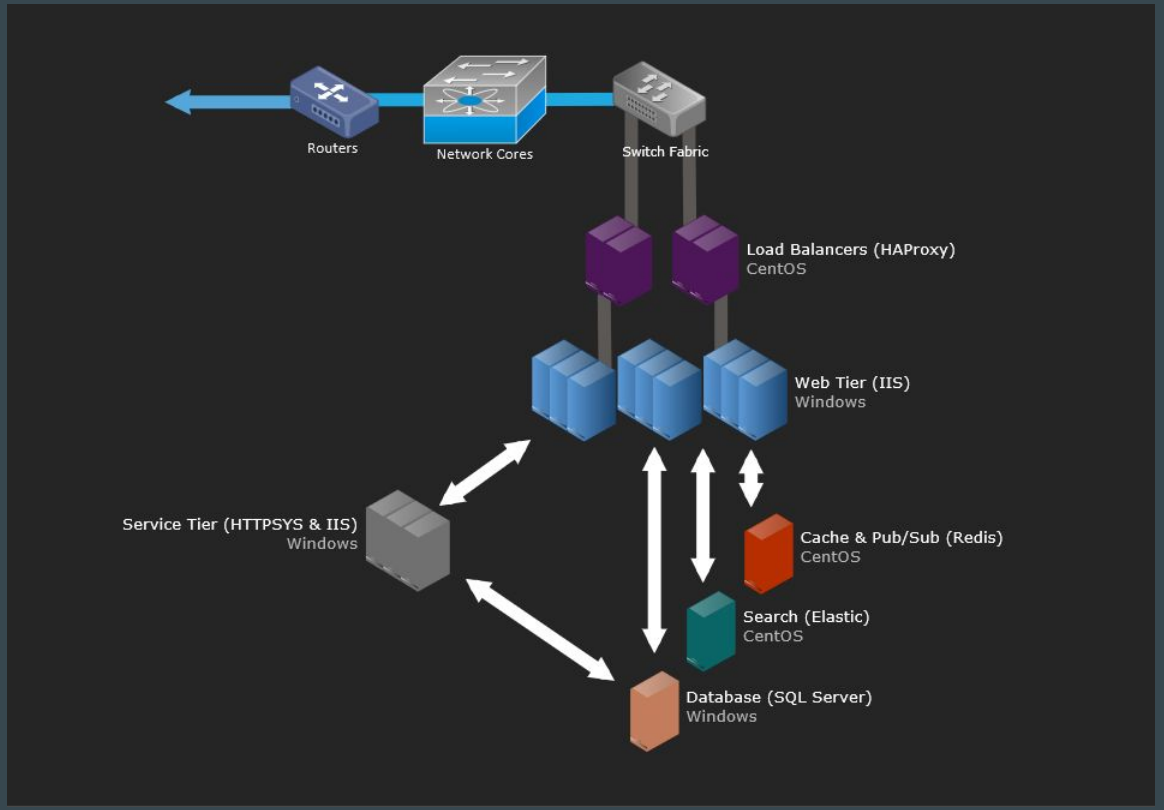
Pros of the Monolith!

- **Legacy** - The majority of enterprise solutions were built using this architecture. They still work and do their job well.
- **“Simple Development” (At first)** - The architecture is well known to any developer. Most frameworks, existing tools, scripts, servers, etc. are compatible. All code in one codebase.
- **“Simple Deployment” (At first)** - Package your entire application and copy to a server.
- **“Simple Testing” (At first)** - The application is deployed all at once which means the features are available as soon as it’s launched. Thus, it can be tested simultaneously and there’s no need to wait for additional dependencies to start testing. Greatly simplifies this process.
- **Single Codebase** - Everything is under one roof. Easy to share with across all stages of the development pipeline.
- **IDE-Friendly** - Plenty of monolithic friendly development environments.

Real World Example of Monolith

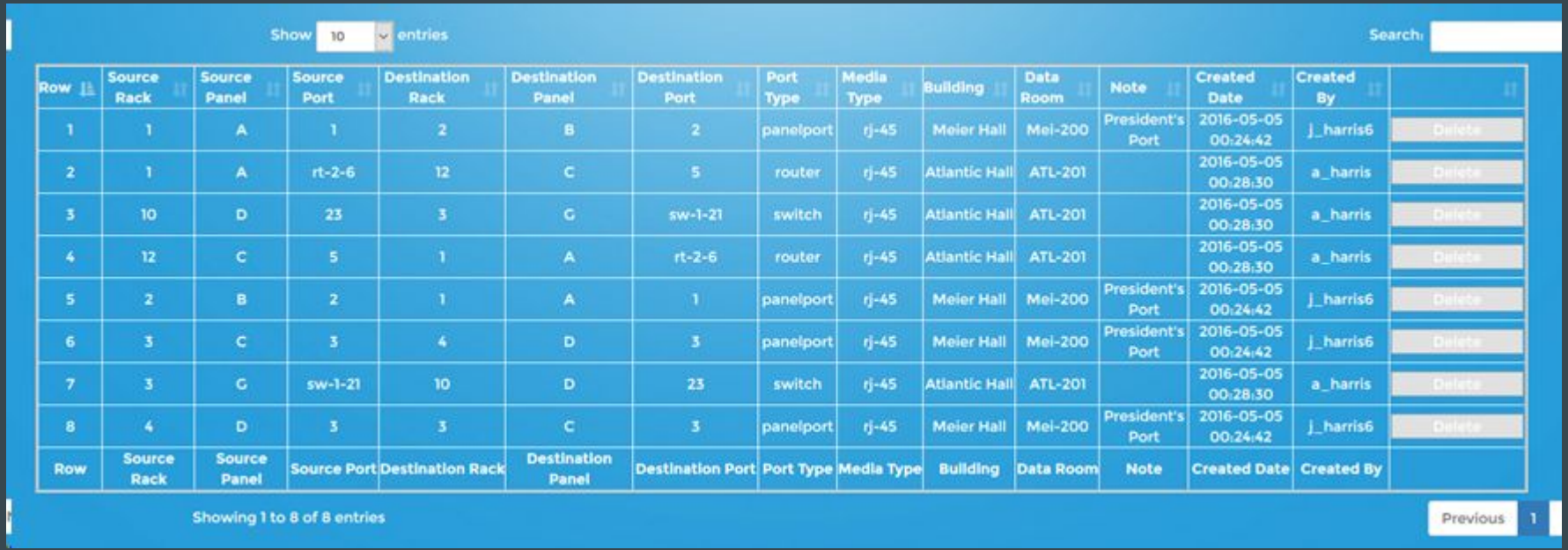
Stack Overflow: The Architecture - 2016 Edition

Stackoverflow



More Examples of the Monolith

Your Computer Science Capstone Projects...usually are monoliths... at least mine was.



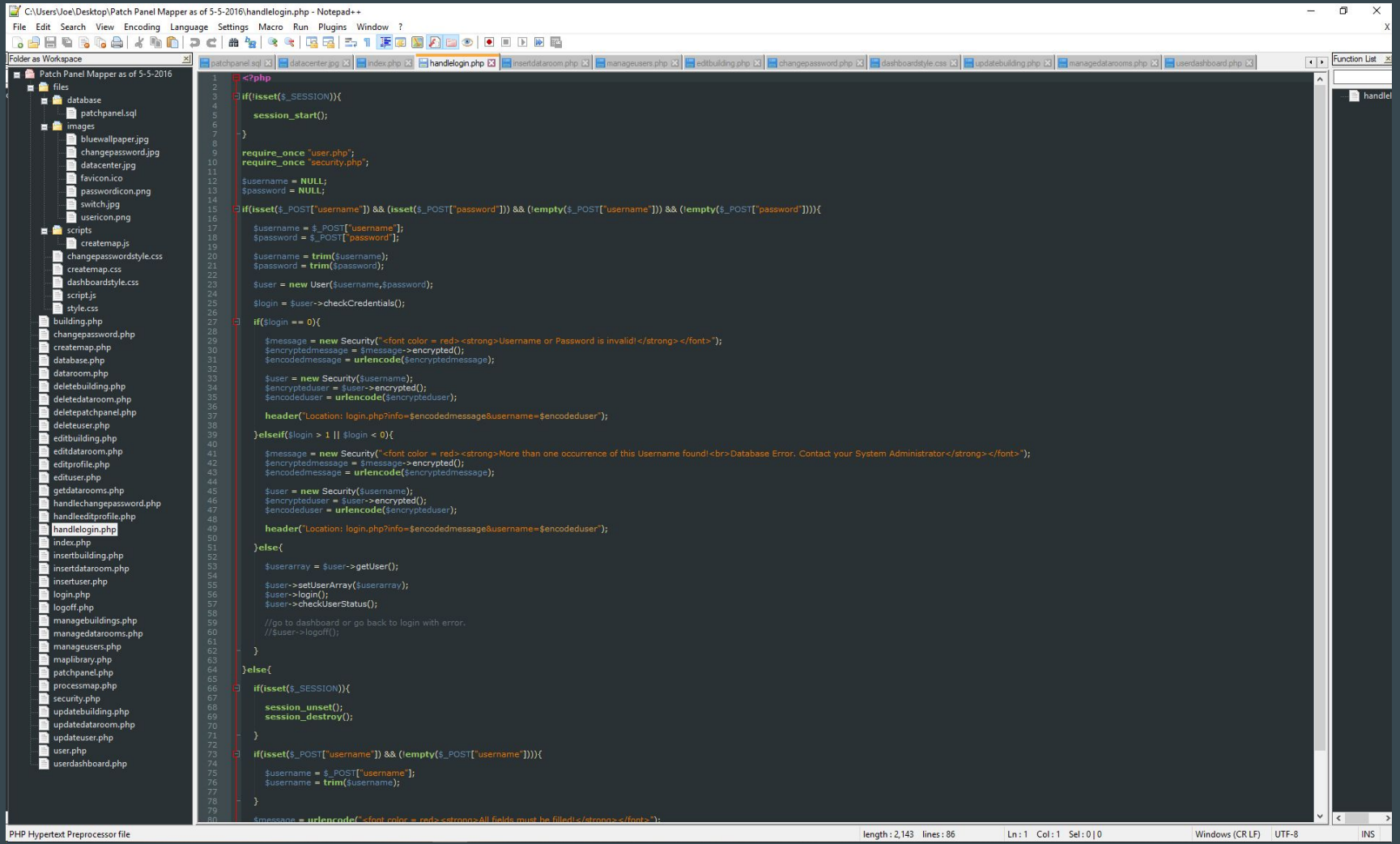
Show 10 entries

Search:

Row	Source Rack	Source Panel	Source Port	Destination Rack	Destination Panel	Destination Port	Port Type	Media Type	Building	Data Room	Note	Created Date	Created By	
1	1	A	1	2	B	2	panelport	rj-45	Meier Hall	Mei-200	President's Port	2016-05-05 00:24:42	j_harris6	Delete
2	1	A	rt-2-6	12	C	5	router	rj-45	Atlantic Hall	ATL-201		2016-05-05 00:28:30	a_harris	Delete
3	10	D	23	3	G	sw-1-21	switch	rj-45	Atlantic Hall	ATL-201		2016-05-05 00:28:30	a_harris	Delete
4	12	C	5	1	A	rt-2-6	router	rj-45	Atlantic Hall	ATL-201		2016-05-05 00:28:30	a_harris	Delete
5	2	B	2	1	A	1	panelport	rj-45	Meier Hall	Mei-200	President's Port	2016-05-05 00:24:42	j_harris6	Delete
6	3	C	3	4	D	3	panelport	rj-45	Meier Hall	Mei-200	President's Port	2016-05-05 00:24:42	j_harris6	Delete
7	3	G	sw-1-21	10	D	23	switch	rj-45	Atlantic Hall	ATL-201		2016-05-05 00:28:30	a_harris	Delete
8	4	D	3	3	C	3	panelport	rj-45	Meier Hall	Mei-200	President's Port	2016-05-05 00:24:42	j_harris6	Delete
Row	Source Rack	Source Panel	Source Port	Destination Rack	Destination Panel	Destination Port	Port Type	Media Type	Building	Data Room	Note	Created Date	Created By	

Showing 1 to 8 of 8 entries

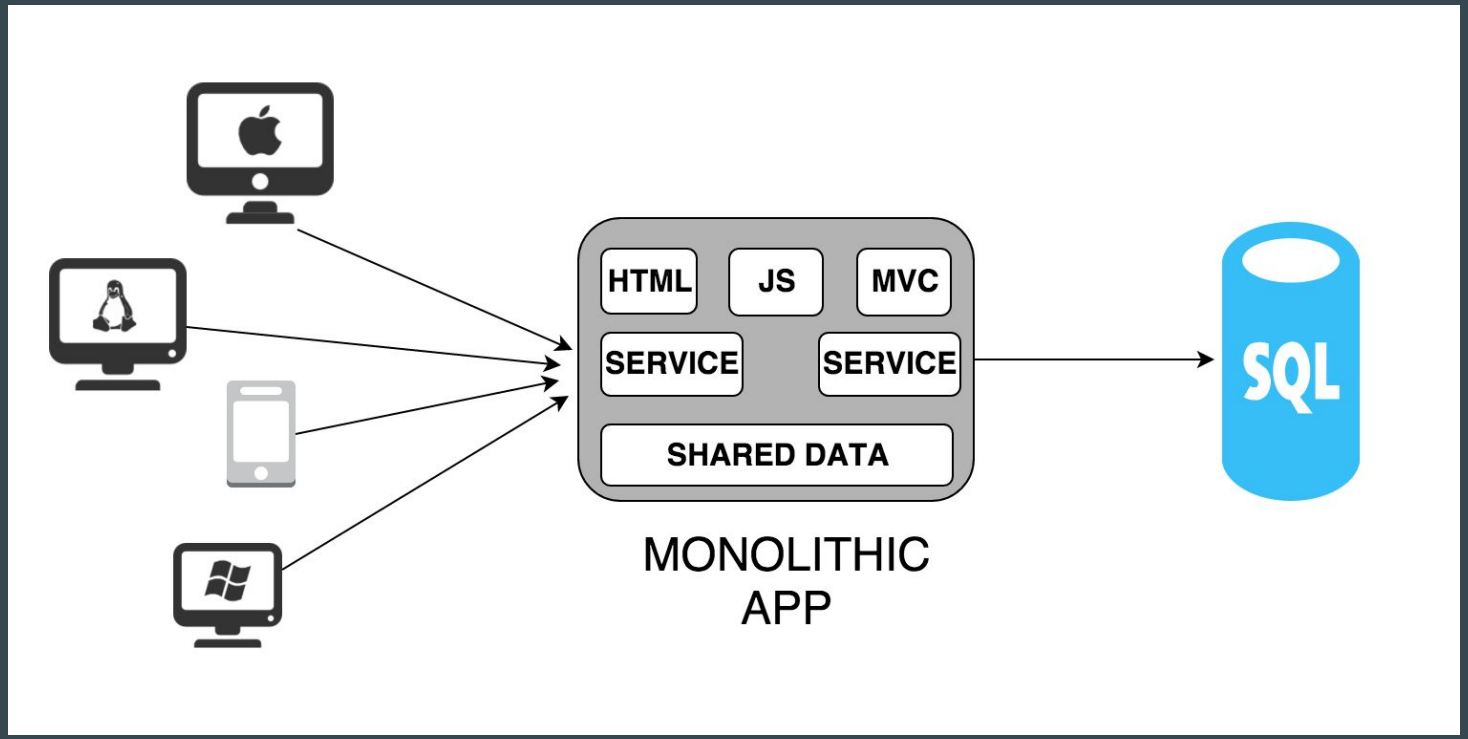
Previous 1


C:\Users\Joe\Desktop\Patch Panel Mapper as of 5-5-2016\handlelogin.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Folder as Workspace
Patch Panel Mapper as of 5-5-2016
files
database
patchpanel.sql
images
bluewallpaper.jpg
changepassword.jpg
datacenter.jpg
favicon.ico
passwordicon.png
switch.jpg
usericon.png
scripts
createmap.js
changepasswordstyle.css
createmap.css
dashboardstyle.css
script.js
style.css
building.php
changepassword.php
createmap.php
database.php
dataroom.php
deletebuilding.php
deletedataroom.php
deletepatchpanel.php
deleteuser.php
editbuilding.php
editdataroom.php
editprofile.php
edituser.php
getdatarooms.php
handlechangepassword.php
handleeditprofile.php
handlelogin.php
index.php
insertbuilding.php
insertdataroom.php
insertuser.php
login.php
logoff.php
managebuildings.php
managedatarooms.php
manageusers.php
maplibrary.php
patchpanel.php
processmap.php
security.php
updatebuilding.php
updatedataroom.php
updateuser.php
user.php
userdashboard.php
patchpanel.sql
datacenter.jpg
index.php
handlelogin.php
insertdataroom.php
manageusers.php
editbuilding.php
changepassword.php
dashboardstyle.css
updatebuilding.php
managedatarooms.php
userdashboard.php
Function List
handlel
<?php
if(!isset($_SESSION)){
session_start();
}
require_once "user.php";
require_once "security.php";
$username = NULL;
$password = NULL;
if(isset($_POST["username"]) && (isset($_POST["password"])) && (!empty($_POST["username"])) && (!empty($_POST["password"]))){
$username = $_POST["username"];
$password = $_POST["password"];
$username = trim($username);
$password = trim($password);
$user = new User($username,$password);
$login = $user->checkCredentials();
if($login == 0){
$message = new Security("Username or Password is invalid!");
$encryptedmessage = $message->encrypted();
$encodedmessage = urlencode($encryptedmessage);
$user = new Security($username);
$encrypteduser = $user->encrypted();
$encodeduser = urlencode($encrypteduser);
header("Location: login.php?info=$encodedmessage&username=$encodeduser");
}elseif($login > 1 || $login < 0){
$message = new Security("More than one occurrence of this Username found!
Database Error. Contact your System Administrator");
$encryptedmessage = $message->encrypted();
$encodedmessage = urlencode($encryptedmessage);
$user = new Security($username);
$encrypteduser = $user->encrypted();
$encodeduser = urlencode($encrypteduser);
header("Location: login.php?info=$encodedmessage&username=$encodeduser");
}else{
$userarray = $user->getUser();
$user->setUserArray($userarray);
$user->login();
$user->checkUserStatus();
//go to dashboard or go back to login with error.
//$user->logoff();
}
}else{
if(isset($_SESSION)){
session_unset();
session_destroy();
}
if(isset($_POST["username"]) && (!empty($_POST["username"]))){
$username = $_POST["username"];
$username = trim($username);
}
$message = urlencode("All fields must be filled!");
PHP Hypertext Preprocessor file
length : 2,143 lines : 86
Ln : 1 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

Cons of the Monolith...

- **High Learning Curve -** New developers on a team have a high learning curve understanding the massive codebase and how everything interconnects.
- **Maintenance** - Coupling between modules causes random bugs when changes are made. Harder to maintain as it grows.
- **Downtime** - Deployments and upgrades require downtime.
- **Barrier to New Tech** - Implementing new technologies could turn into a huge problem since it will affect the entire system.
- **Crash Impact Radius** - If one part of the system crashes, the entire system crashes.
- **Poor Vertical Scaling -** Installing new hardware or upgrading hardware to an existing monolith is a nightmare. Imagine upgrading SSDs that contains the codebase...

Parts of a Monolithic Application



How it can sometimes feel to work on Monolithic Applications

Every change affects the entire application. The larger the codebase the more daring the act of modifying code becomes.

Key Features of Microservices

- Code of each microservice is stored in an isolated *container**, runs its own memory space, and functions independently.
- Communication between services is a bit more complex since developers should use logic to connect them and deal with failures that sometimes occur.
- Scaling of one component is possible.
- Clearly organized architecture. Decoupled units have their specific jobs, can be reconfigured without affecting the entire application.

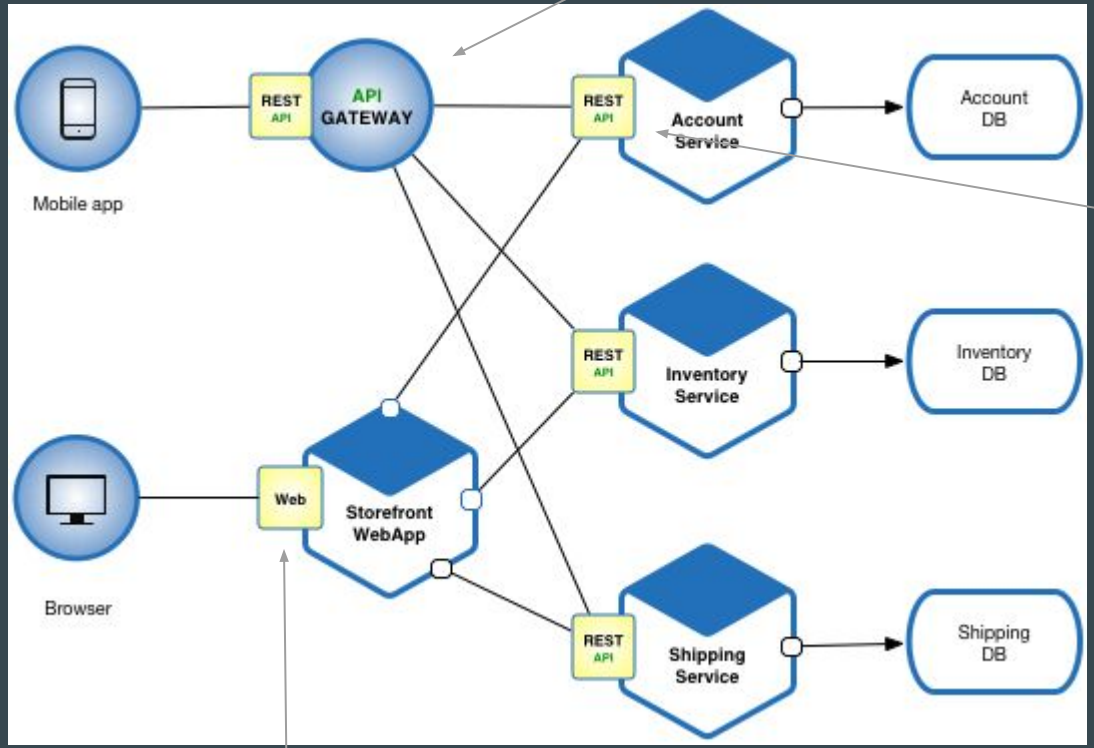
* We will revisit containers.

Pros of Microservices!

- Code complexity greatly reduced.
- Service separation promotes decoupled designs that have less bugs.
- Less of a learning curve to be productive (you don’t need to study a giant codebase).
- Deployments don’t require downtime.
- If a microservice crashes, the rest of the system keeps going.
- Each microservice can be scaled individually according to its needs.
- Services can use different tech stacks (developers are free to code in any language).


....MANY
SERVICES THAT
COMMUNICATE ?
http://www.raptorsrepublic.com/

Separate API Gateway for mobile apps are quite common as well.
Why have the same API for mobile and nonmobile?



Very common for each service to have its own database.

The communication interface for each service is an API.
Less IPC OS communication.

UI accessible via web.

Note: All the communication between services and the API Gateway are done entirely over HTTP(S).

Real World Example of Microservices

[Microservices at Netflix Scale 2016]

August 2008

NETFLIX Your Account | Queue | Help

We're Sorry DVD Shipments Are Delayed

Dear Betsy,

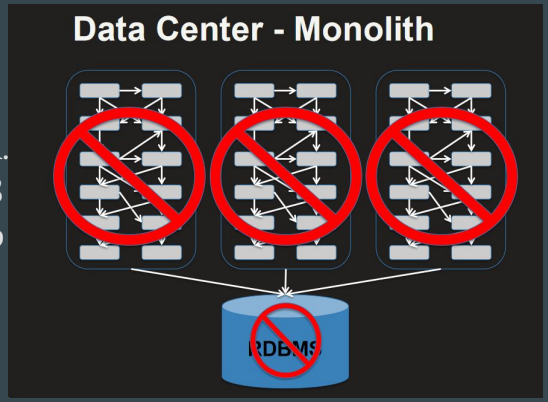
Our shipping system is unexpectedly down. We received a DVD back from you and should have shipped you a DVD, but we likely have not. Our goal is to ship DVDs as soon as possible, and we will keep you posted on the status of your DVD shipments.

We are sorry for any inconvenience this has caused. If your DVD shipment is delayed, we will be issuing a credit to your account in the next few days. You don't need to do anything. The credit will be automatically applied to your next billing statement.

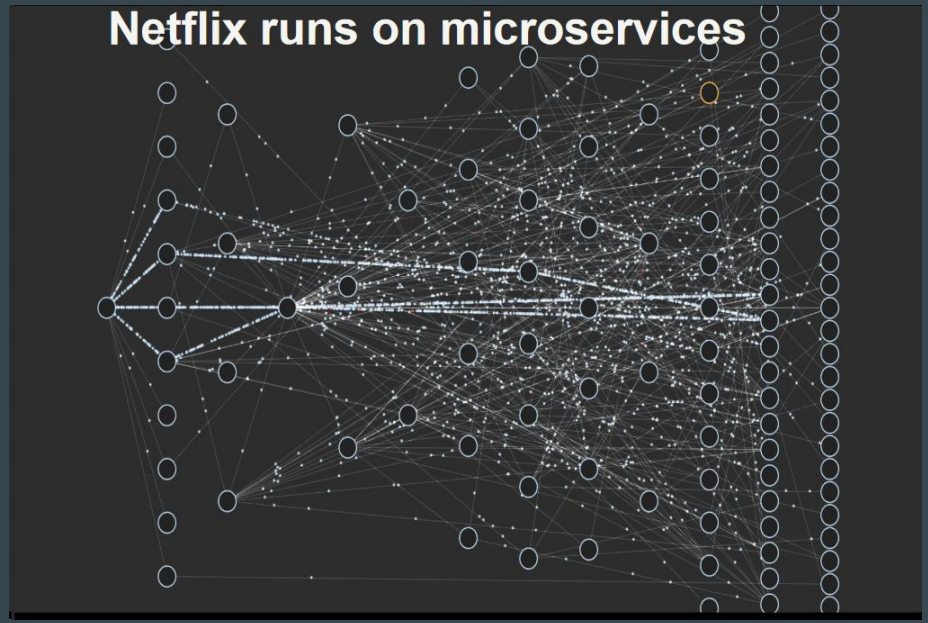
Again, we apologize for the delay and thank you for your understanding. If you need further assistance, please call us at 1-888-638-3549.

-The Netflix Team

Netflix was originally a Monolithic Application. In 2008 they started a 8 year process to move to Microservices in AWS.



Netflix is the world's leading Internet television network with over 81 million members in over 190 countries enjoying more than 125 million hours of TV shows and movies per day, including original series, documentaries and feature films.



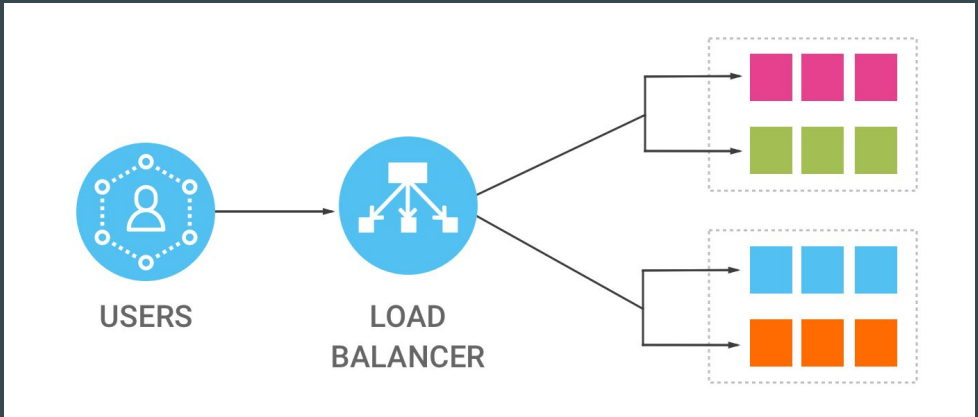
Cons of Microservices...

- Complexity moves from the code to the interactions between services. Example: Microservices run in a separated processes and communicate over the network and a dedicated mechanism (like REST calls or messaging). Making HTTP requests (via a client API) and processing it is more laborious than doing a simple programmatic method call.
- Complex database joins must be implemented by the application. Can cause latency.
- Deployments have a lot of moving pieces.
- Lower performance when a request “pinballs” through multiple microservices.
- Testing the whole application can be more tedious compared to a Monolithic application due to the different services.
- Management of multiple databases, codebases, and transactions can be very cumbersome.
- Network communication can be highly variable. HTTP Request/Responses dropping, getting lost, etc. *Good luck troubleshooting!*
- We have to come up with a service registry that allows microservices to find each other otherwise there can be no communication.

Parts of a Microservice Application

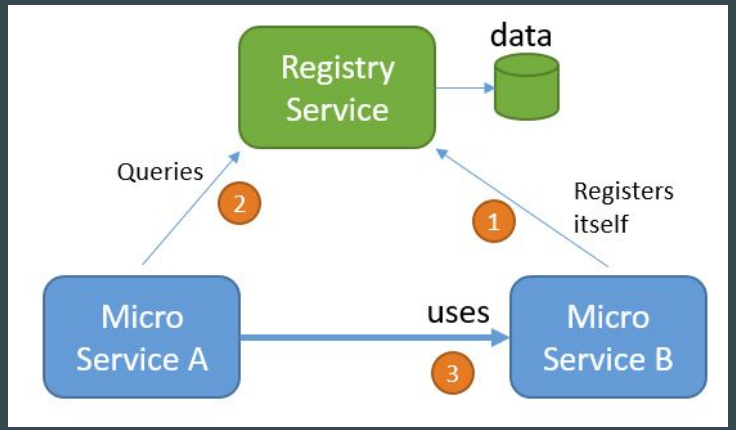
Load Balancer

- All Microservices are accessed through the Load Balancer.
- Microservices come and go but the Load Balancer is the “switchboard”.
- Enables horizontal scaling of services.
- Makes Rolling upgrades possible.



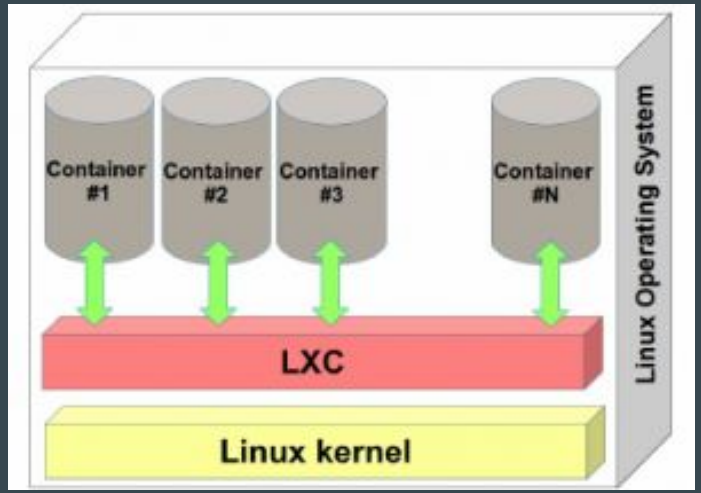
Service Registry

- Datastore that keeps a list of running services.
- Must be redundant, highly available, and fast.
- Services make themselves known to the registry when they start.
- They are removed when they end or crash.
- The Load Balancer is dynamically reconfigured when the registry changes.



Containers

- Make services portable across host platforms.
- Provide an additional layer of isolation over processes.
- Allow each service to use its own dependencies.
- Simplify managing of network ports.

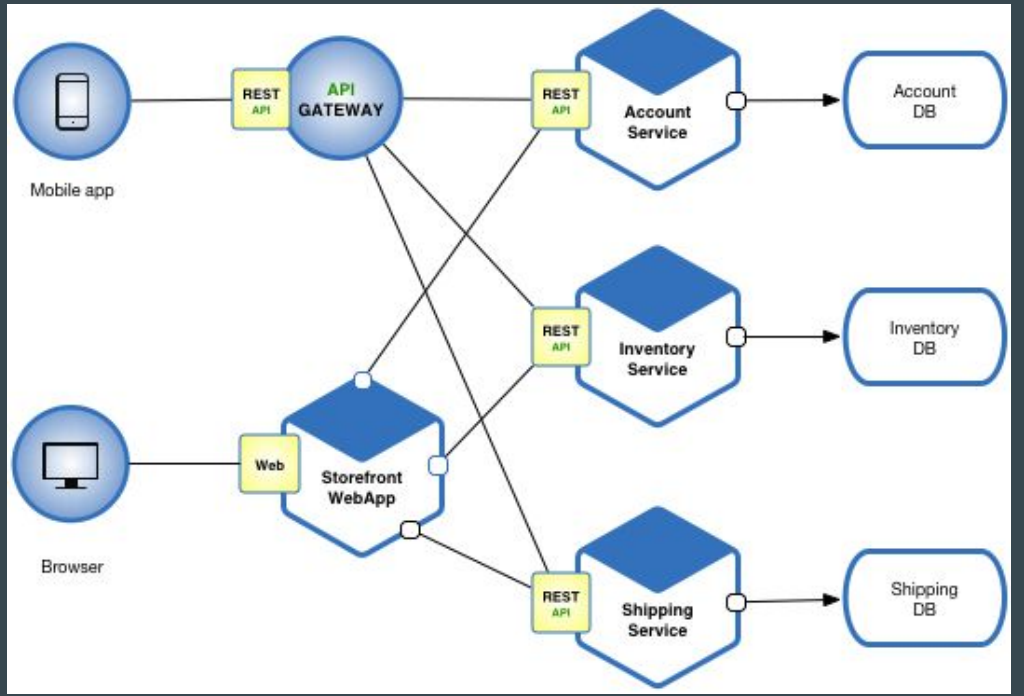


Storage

- Service registry, databases, message queries, etc. are stateful services.
- It is important to make these services robust in order to prevent data loss.
- Most storage solutions have clustering options.

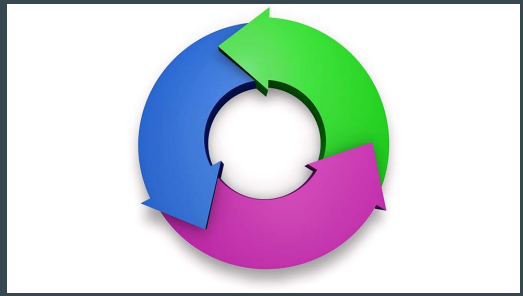
Application Microservices

- Services that a developer writes.
- Should be stateless.
- They can start and stop at any time, without data loss.
- Horizontally scalable.
- Can be written as simple web APIs using any language and framework (RPC can be used to receive requests as well).



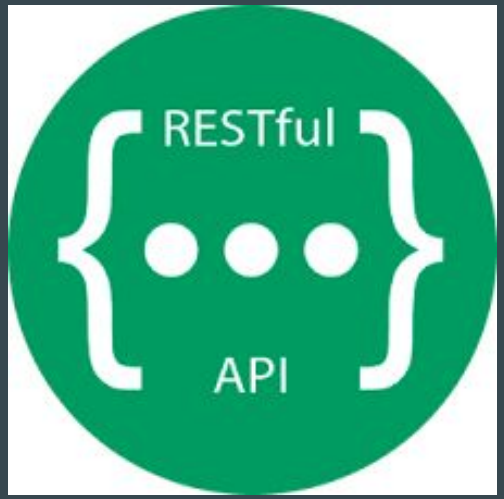
Lifecycle of a Microservice

- On startup the Microservice registers with the Service Registry.
- The Load Balancer detects the change in the registry and updates itself.
- The new service starts receiving traffic from the Load Balancer.
- If more than one instance of the service exists (in the case of a horizontal scale) then the traffic is split among the instances.
- The service sends “keep alive” signals, or responds to periodic health checks.
- When the service is stopped, or stops sending keep alives, or fails a health check, then it is removed from the registry as well as the Load Balancer.



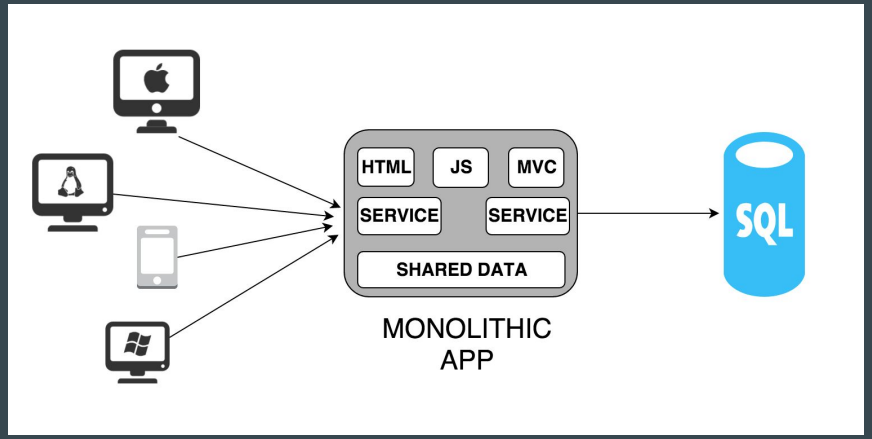
Service to Service Communication

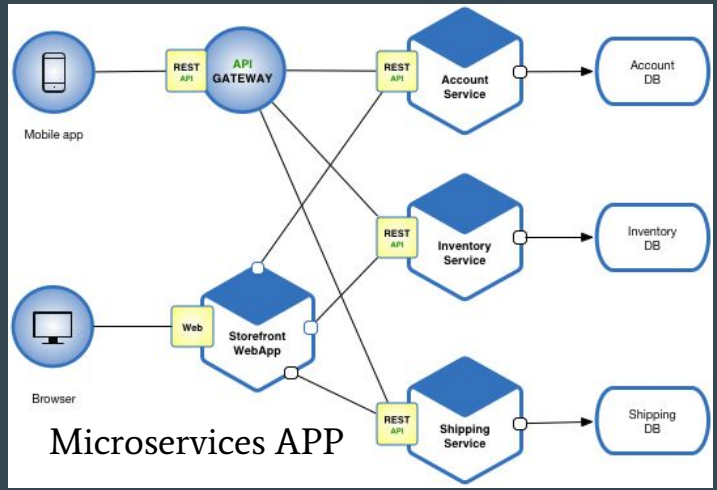
- Outside clients connect over HTTP/REST/WebSocket.
- The service receiving the client request may need to invoke other services.
- Services communicate with each other using HTTP or REST.
- Payloads should use well known formats: JSON, protobufs, etc.



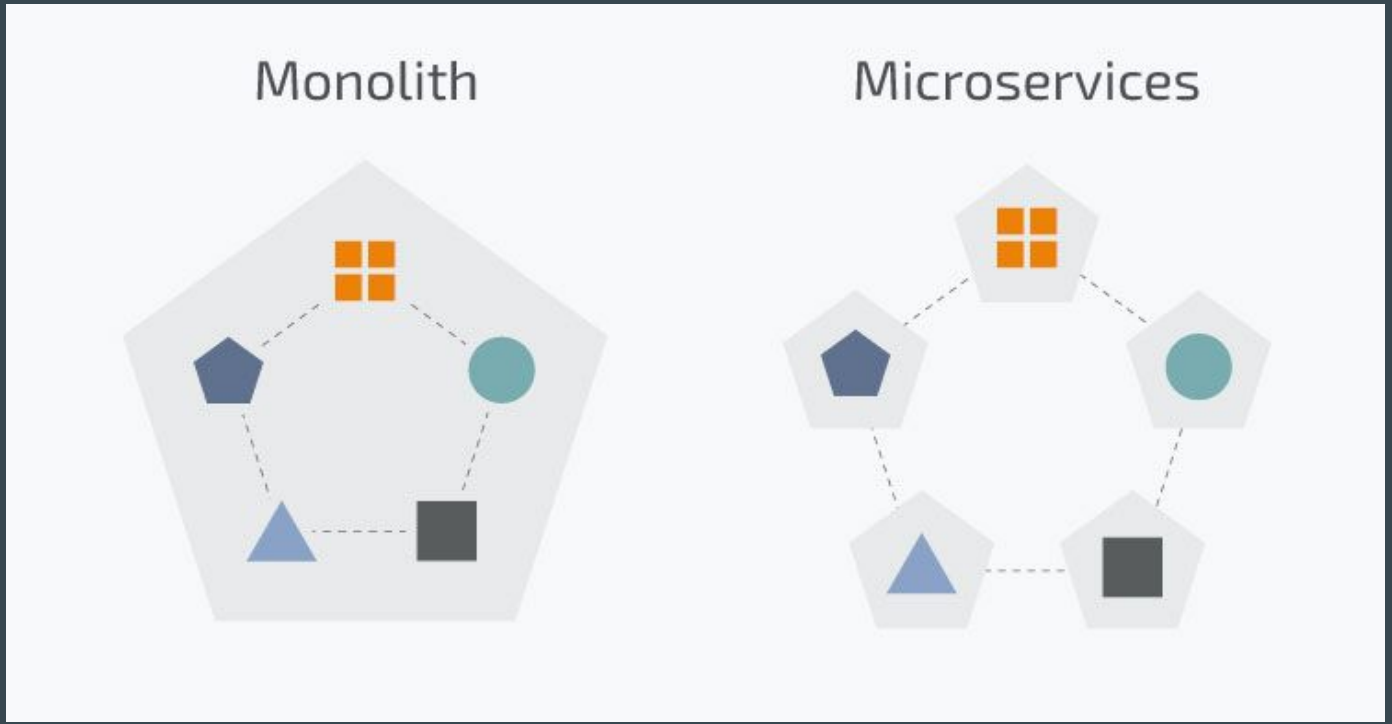
Quick Overview

	Microservices	Monolithic
Pros	• Variety • Easy to understand and develop • Scalability • Reliability • Adoption of new technologies • Flexible teams	• Simple development • Simple deployment • Simple testing • Simple sharing • IDE-friendly
Cons	• Difficult integration testing • Operational complexity • Time consuming • Skills demand	• Difficult to understand and change • Limited agility • Barrier to new technologies • Barrier to continuous delivery/deployment/integration • Slowness • Poor reliability
Price	• Initial costs are higher • Can reduce costs while operating	• Initial costs are lower • Can increase costs while operating





So... Which Architecture is Better?



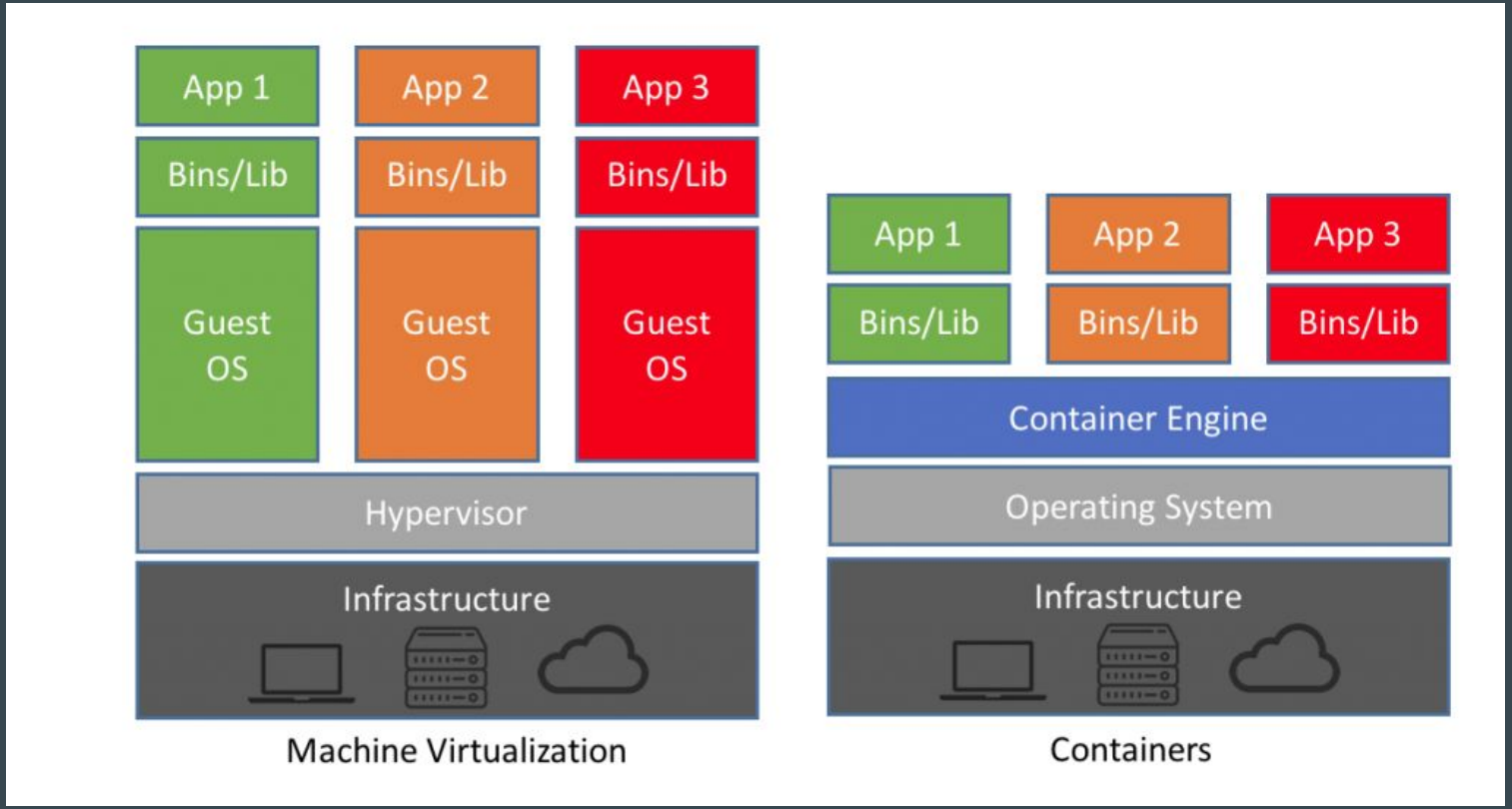

Let's Back Up...

How Does the Cloud Fit into This?

Turns out Cloud Computing is slightly important to this subject...

One quick thing before we talk about the ALMIGHTY CLOUD....

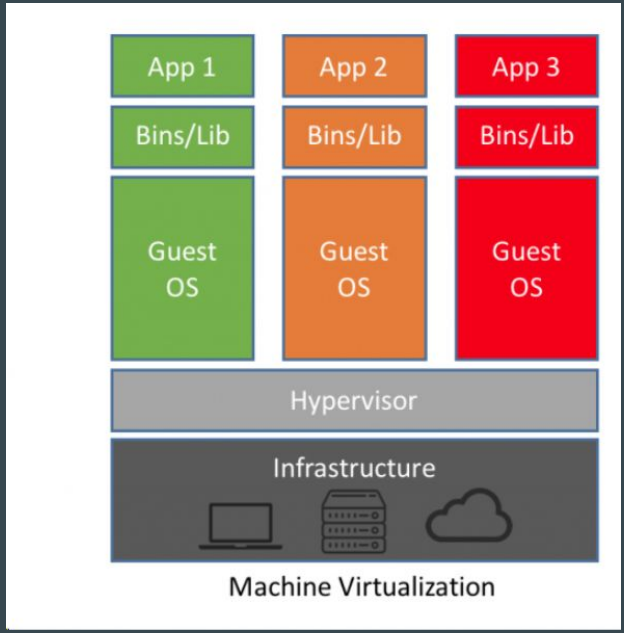
Hypervisors vs. Containers

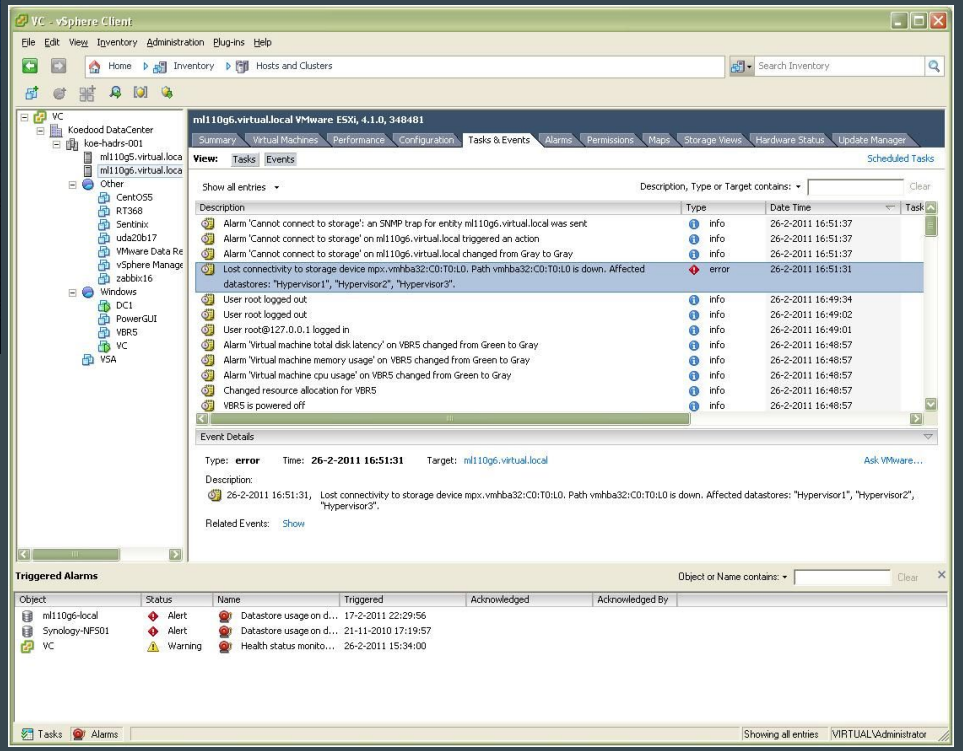


Hypervisors

- A Hypervisor is software that allows a physical machine to host multiple virtual guest operating systems.
- Each Guest OS is completely segregated from the Host OS and the other Guest OSes.
- Shared Hardware.
- No shared processes.
- Each provisioned space for the Guest

  requires an entirely separate OS from the Host.





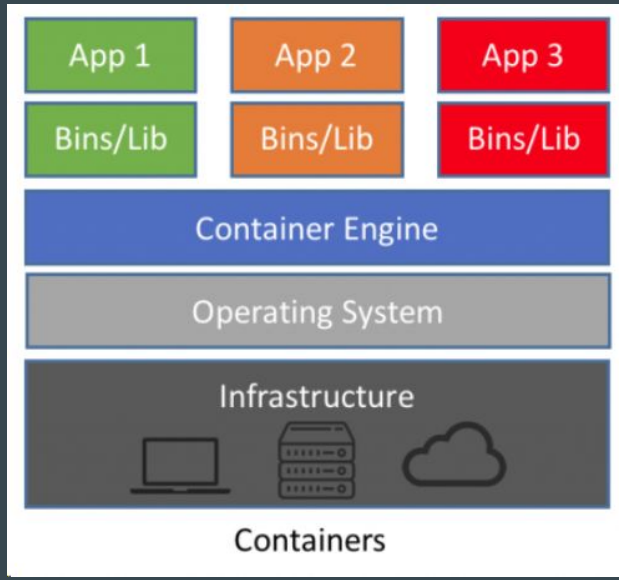
vmware
vSphere

LXC

- LXC = LinuX Containers.
- Isolate applications and its Binaries and Libraries but also allow access to the host Linux Kernel.
- Kernel namespaces (ipc, uts, mount, pid, network, user)
- This allows containers to be completely portable.
- OS level virtualization method for running multiple

  isolated Linux Systems (Containers) on a control host

  using a single Linux Kernel.



```
root@workbase ~
# docker ps                                                                                                    127 ↵
CONTAINER ID        IMAGE                          COMMAND                  CREATED
   STATUS              PORTS                                    NAMES
672175da3a51        nginx                          "nginx -g 'daemon of…"   6 hours ago
   Up 6 hours          80/tcp                                   nginx-bg
2eeee33f09ce        ansible/awx_task:latest        "/tini -- /bin/sh -c…"   3 months ago
   Up 18 hours         8052/tcp                                 awx_task
c64666a466d0        ansible/awx_web:latest         "/tini -- /bin/sh -c…"   3 months ago
   Up 18 hours         0.0.0.0:8000->8052/tcp                   awx_web
72d3c938c945        memcached:alpine               "docker-entrypoint.s…"   3 months ago
   Up 18 hours         11211/tcp                                memcached
b2ea65cf0995        rabbitmq:3                     "docker-entrypoint.s…"   3 months ago
   Up 18 hours         4369/tcp, 5671-5672/tcp, 25672/tcp       rabbitmq
17a696f2c856        postgres:9.6                   "d
   Up 18 hours         5432/tcp
root@workbase ~
#
```

```
[root@tecmint ~]# docker run -it ubuntu bash
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.1  18112  1972 ?        Ss   06:58   0:00 bash
root        15  0.0  0.1  15512  1144 ?        R+   06:58   0:00 ps aux
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# uname -a
Linux ff3ccd9fb856 2.6.32-573.12.1.el6.x86_64 #1 SMP Tue Dec 15 21:19:08 UTC 201
5 x86_64 x86_64 x86_64 GNU/Linux
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# w
 06:58:47 up 43 min,  0 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# cat /etc/de
debconf.conf    default/        depmod.d/
debian_version  deluser.conf
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# cat /etc/de
debconf.conf    default/        depmod.d/
debian_version  deluser.conf
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# cat /etc/debian_version
jessie/sid
;root@ff3ccd9fb856: /root@ff3ccd9fb856:/# exit
exit
[root@tecmint ~]# _
```

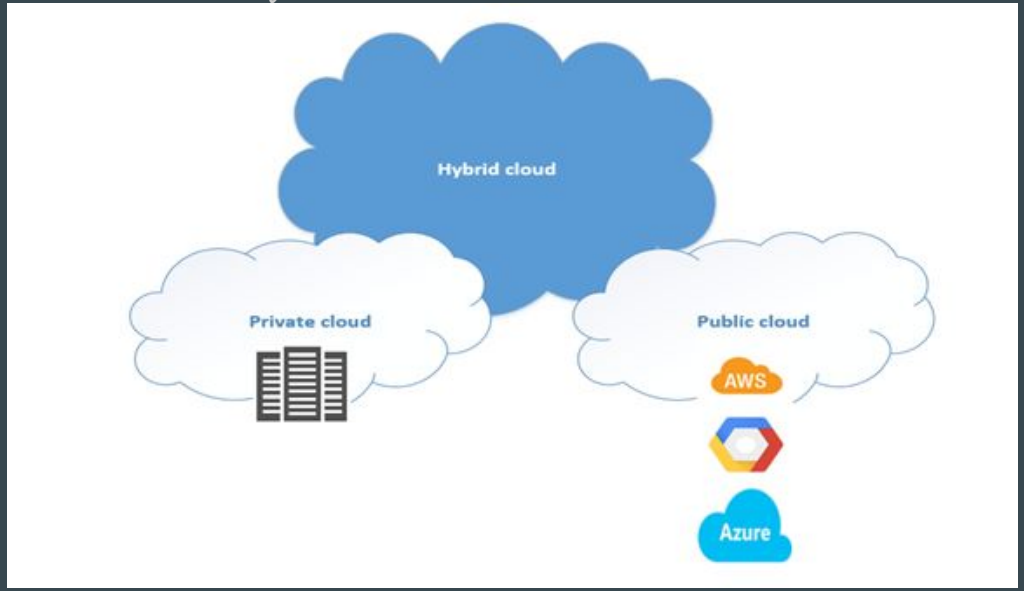
Now we can talk about THE ALMIGHTY CLOUD!

How Does the Cloud Fit into This?

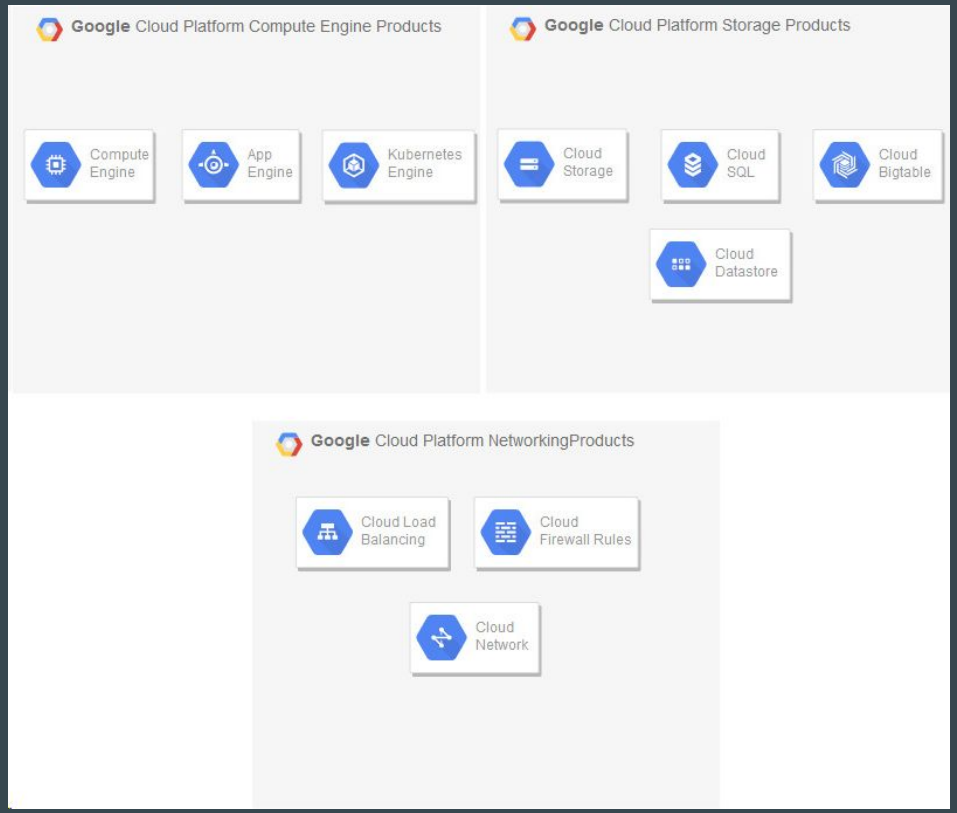
Turns out Cloud Computing is slightly important to this subject...

Cloudy Disclaimer

I am going to focus on the Public Cloud (*Google Cloud Platform being the best*) but it is worth mentioning that it is possible to run Monolithic Applications or Microservices Applications on a Private Cloud or Hybrid Cloud. Many people think of The Cloud as ONLY the Public Cloud. I just want to mention that this is not the case.

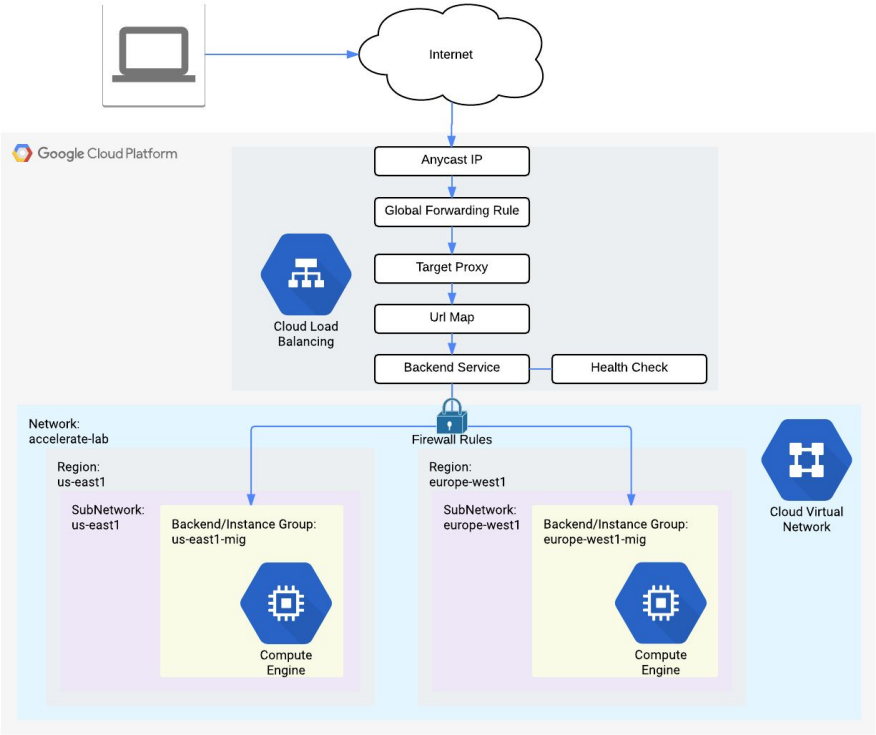


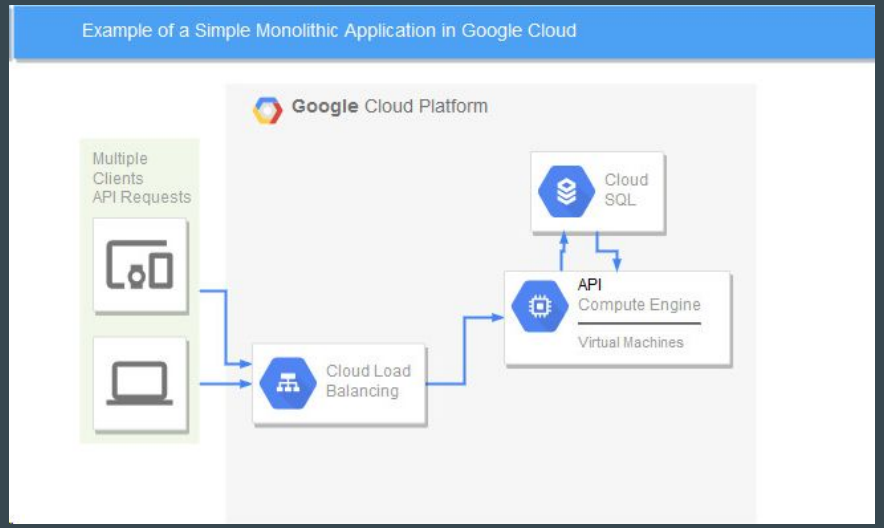
Cloud Technologies



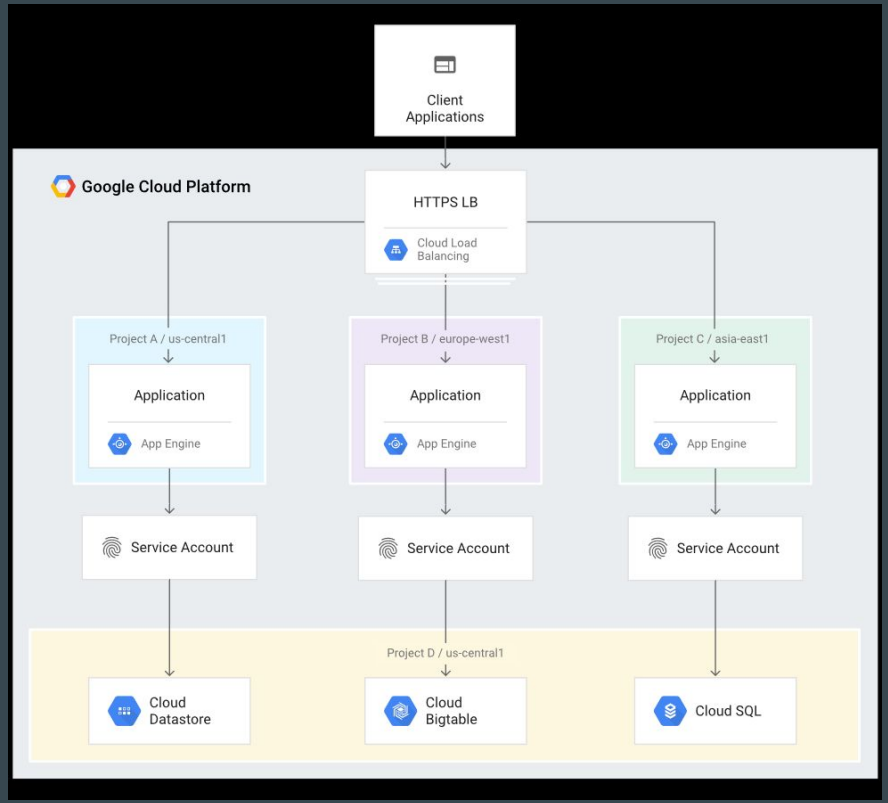
Google Cloud

Monolithic Application in the Cloud Examples

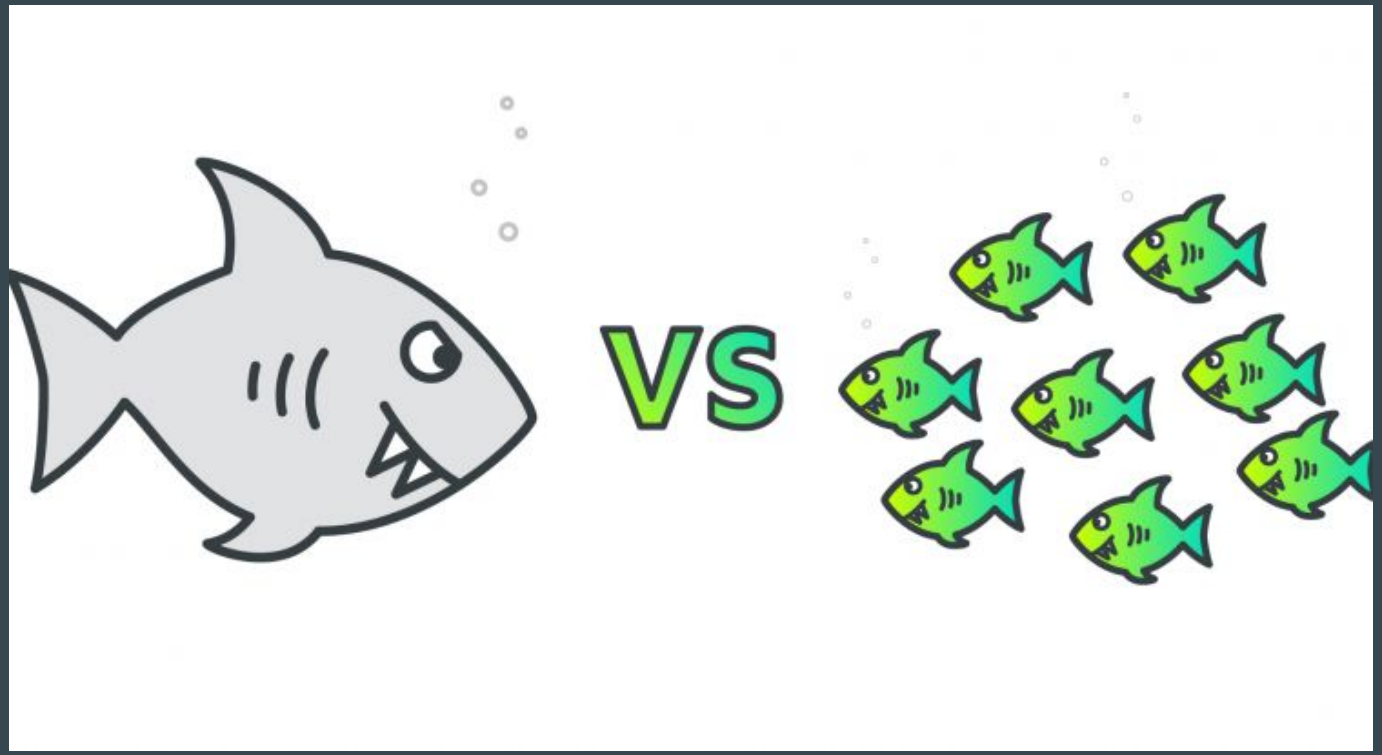




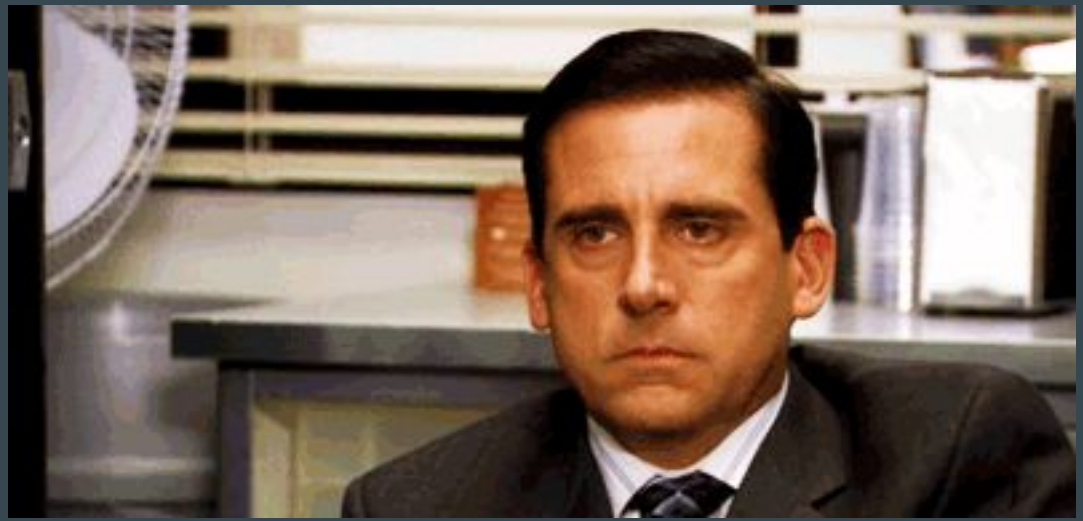
Microservices Application in the Cloud Example



Which Architecture is Better???!!!



My Conclusion is: ***It Depends.***

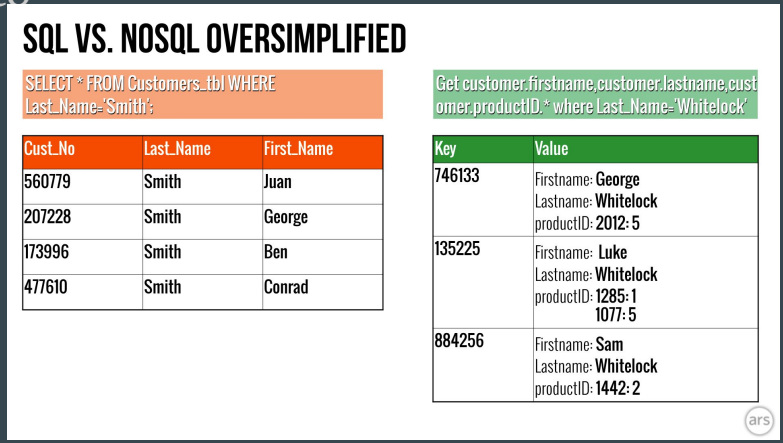


Parting Thoughts

The Monolithic Architecture and Microservices Architecture Debate reminds me of the SQL and NoSQL Debate. NoSQL didn't and will never replace SQL. The relational model will always be useful depending on what problem you are trying to solve. Just as Microservices won't replace Monoliths just because it is the cool new thing. Why go through the trouble of creating a Microservices Application, Pay for the Cloud Hosting, and have a DevOps Team on standby just to

host a small web application?

ROI = Return On Investment



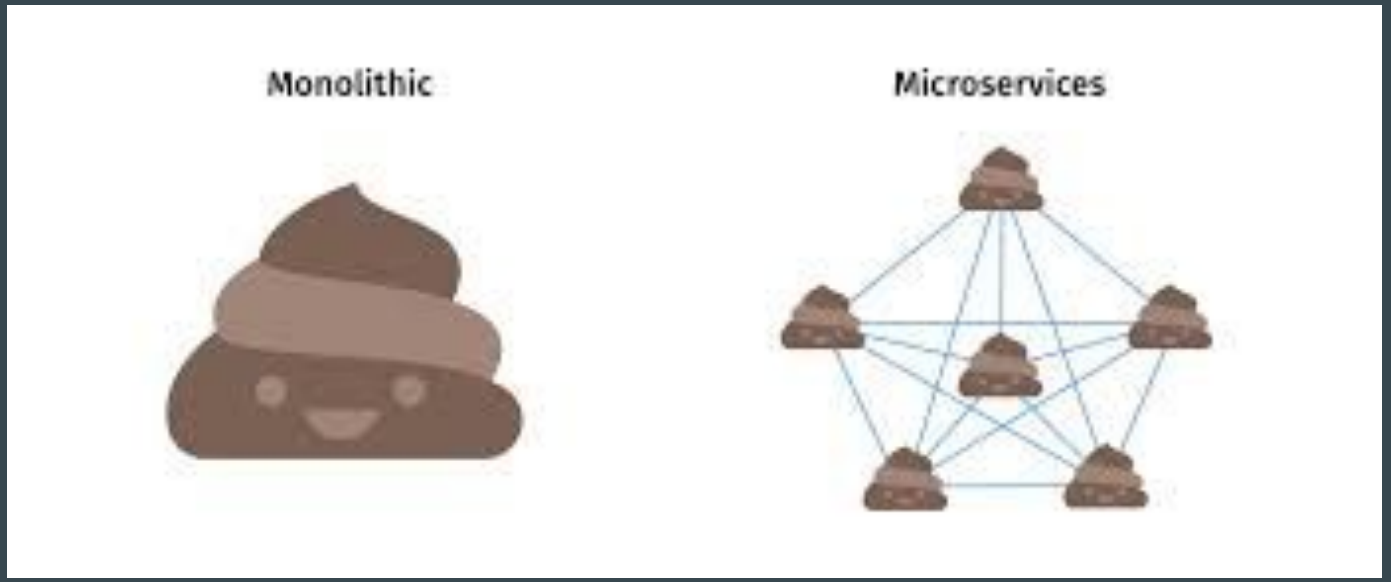
In Conclusion

Monoliths and Microservice Applications offer pros and cons over one another but are not necessarily “better” than its counterpart.

It simply depends if you are planning to write an application that will require the benefits of the Monolith or Microservices for today and into the future.

Final Take-away

None of this matters if you don’t design your code, plan ahead, and understand the technologies. You’ll just end up with Monolithic crap or Micro craps.



"ANY QUESTIONS?"