Table of Contents

Scalable Microservices with Kubernetes

Version

Author	Version	Last update	Comment
luwenwu@cn.ibm.com	V1.0	03/14/18	Create the document

About this Course

https://www.udacity.com/course/scalable-microservices-with-kubernetes--ud615

This course is designed to teach you about managing application containers, using Kubernetes. We've built this course in partnership with experts such as Kelsey Hightower and Carter Morgan from Google and Netflix's former Cloud Architect, Adrian Cockcroft (current Technology Fellow at Battery Ventures), who provide critical learning throughout the course.

Mastering highly resilient and scalable infrastructure management is very important, because the modern expectation is that your favorite sites will be up 24/7, and that they will roll out new features frequently and without disruption of the service. Achieving this requires tools that allow you to ensure speed of development, infrastructure stability and ability to scale. Students with backgrounds in Operations or Development who are interested in managing container based infrastructure with Kubernetes are recommended to enroll!

In this course you will learn to:

Containerize an application by creating Docker config files and build processes to produce all the necessary Docker images
Configure and launch an auto-scaling, self-healing Kubernetes cluster
Use Kubernetes to manage deploying, scaling, and updating your applications
Employ best practices for using containers in general, and specifically Kubernetes, when architecting and developing new Microservices

What You Will Learn

LESSON 1 Introduction to Microservices

Learn how 12-factor apps and the microservices design pattern make modern applications easier to deploy and maintain.

LESSON 2 Building the Containers with Docker

Use Docker to build container images that package an application and its dependencies for deployment on a single machine.

LESSON 3 Kubernetes

The infrastructure to support an application at scale is as important as the application itself.

See how Kubernetes allows you to focus on the big picture.

LESSON 4 Deploying Microservices

Go beyond the theoretical concepts and try out Kubernetes so that you can use it to manage real world apps.

Lesson 1: Introduction to Microservices

Resources

Throughout this course we'll link a lot of different websites and resources to check out. This way, no matter how you learn, you'll always have resources for digging deeper on a specific subject.

Below we have compiled a list of some of our favorites.

Resources:

People

Kelsey Hightower **@kelseyhightower**
Carter Morgan **@_askcarter**
Adrian Cockcroft **@adrianco**
Gundega Dekena **@pytonc**

Books

Kubernetes: Up and Running, Kelsey Hightower The definitive book on Kubernetes. This has been a great resource while making this course.
Building Microservices: Defining Fine-Grained Systems This is the book Kelsey reads before giving talks about microservices. It's that good.

Articles

Martin Fowler on the **Pros** and **Cons of Microservices**
12-Fractured Apps - One of Carters favorites articles where Kelsey breaks down problems with many modern apps and how 12-factor app methodology solves those technical woes.

Tim O'Reilly, **"Open Data: Small Pieces Loosely Joined"** For the history nerds: Quite possibly the first article about Microservices Architecture (before it even had a name).

Videos

Adrian Cockroft **"The Evolution of Microservices"**
Adrian Cockroft **"The State of the Art in Microservices"** (docker specific)
Martin Fowler **"Microservices"** at goto
Craig McLuckie **"The Next Chapter in Native Cloud Computing"** on cloud-native computing as being: container-packaged, dynamically-scheduled, and microservices-oriented

Tools we use in the course

The Go Programming Language

Our app is written in Go. If you're not already using Go, you owe it to yourself to try it out.
https://golang.org/

Google Cloud Shell

A free temp VM preloaded with the tools need to manage our clusters.
https://cloud.google.com/shell/docs/

Docker

We use Docker to package, distribute, and run our application.
https://www.docker.com/

Kubernetes

Once we have an application, we use Kubernetes to handle the heavy lifting of managing, deploying, and scaling our application.
http://kubernetes.io/

Google Container Engine (GKE)

GKE is a hosted Kubernetes service
https://cloud.google.com/container-engine/

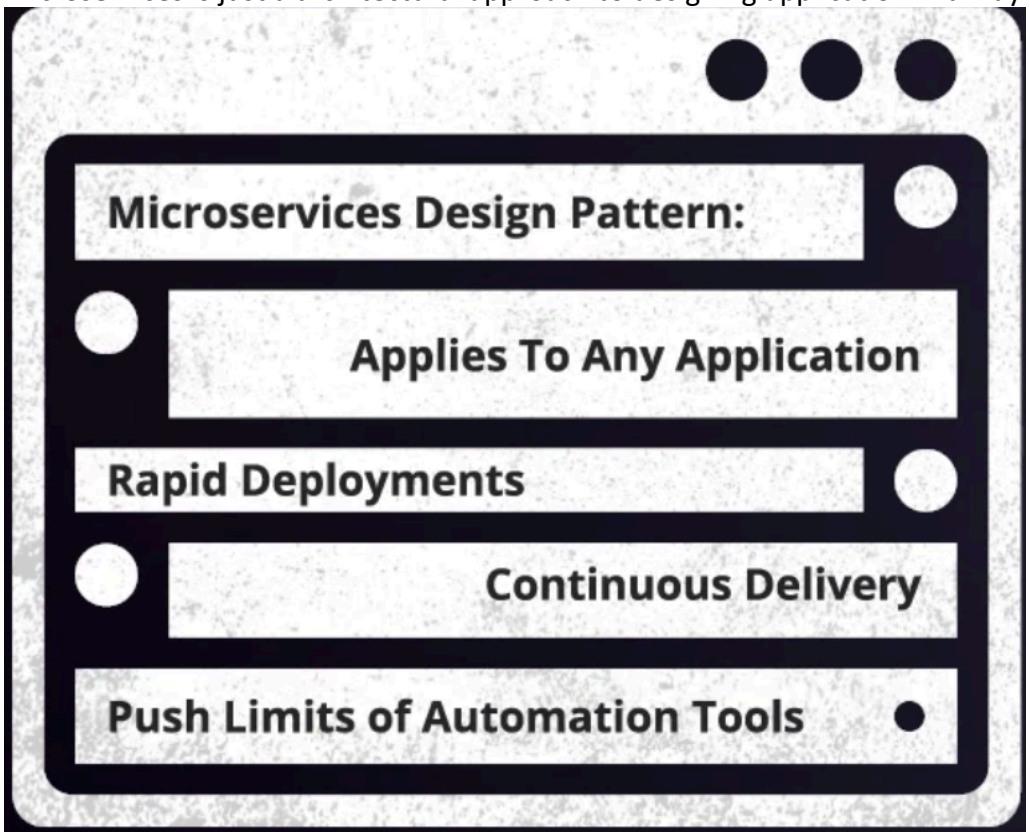
The Evolution of Applications

Adrian Cockcroft:
Traditional application takes time to build and deploy

Recently, takes seconds to deploy by using containers
Microservices is really freeing up everybody to run on their own speed

Microservices

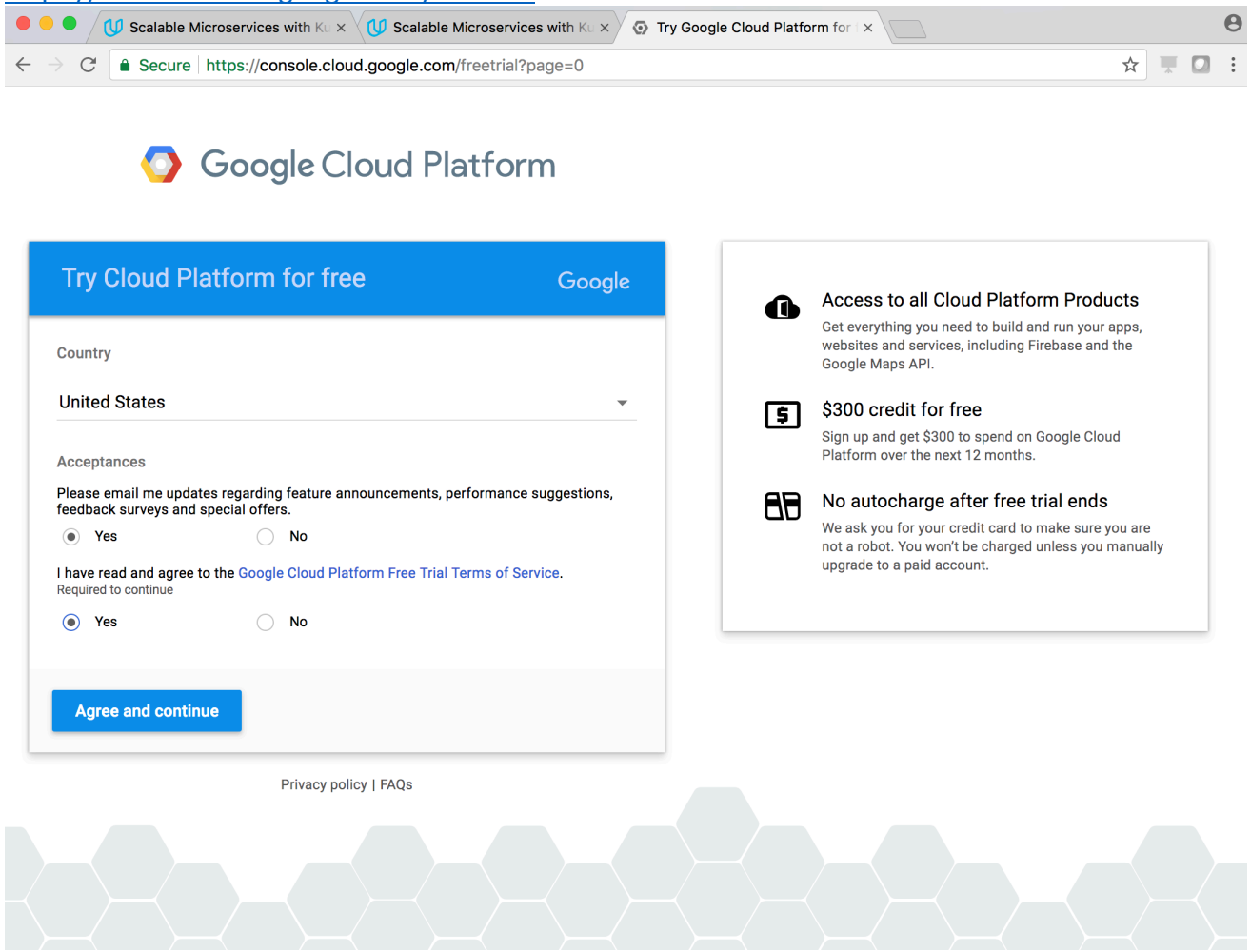
Kelsey Hightower:
Microservices is just a architectural approach to designing application in a way that's modular, easy to deploy, and scale independently.

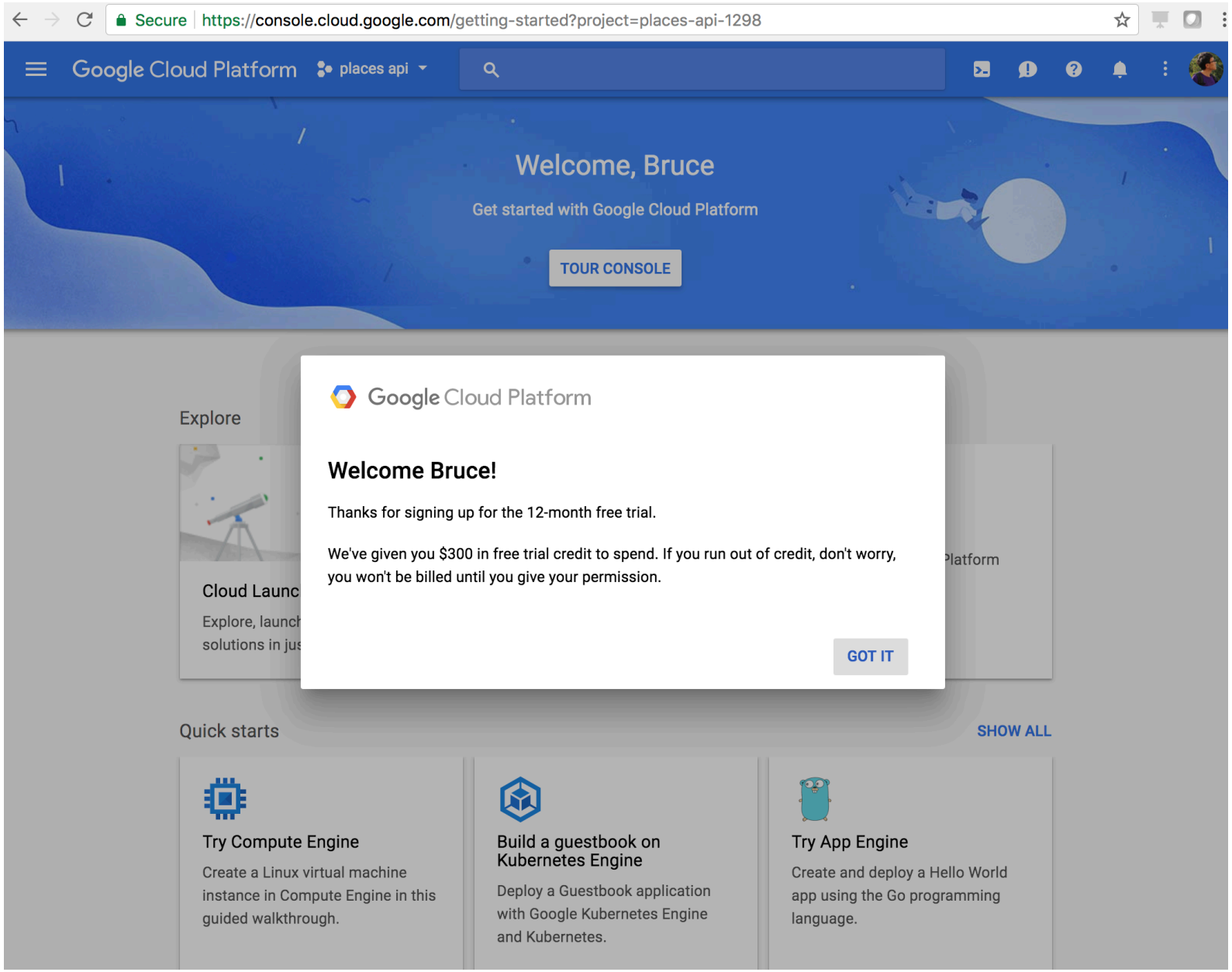


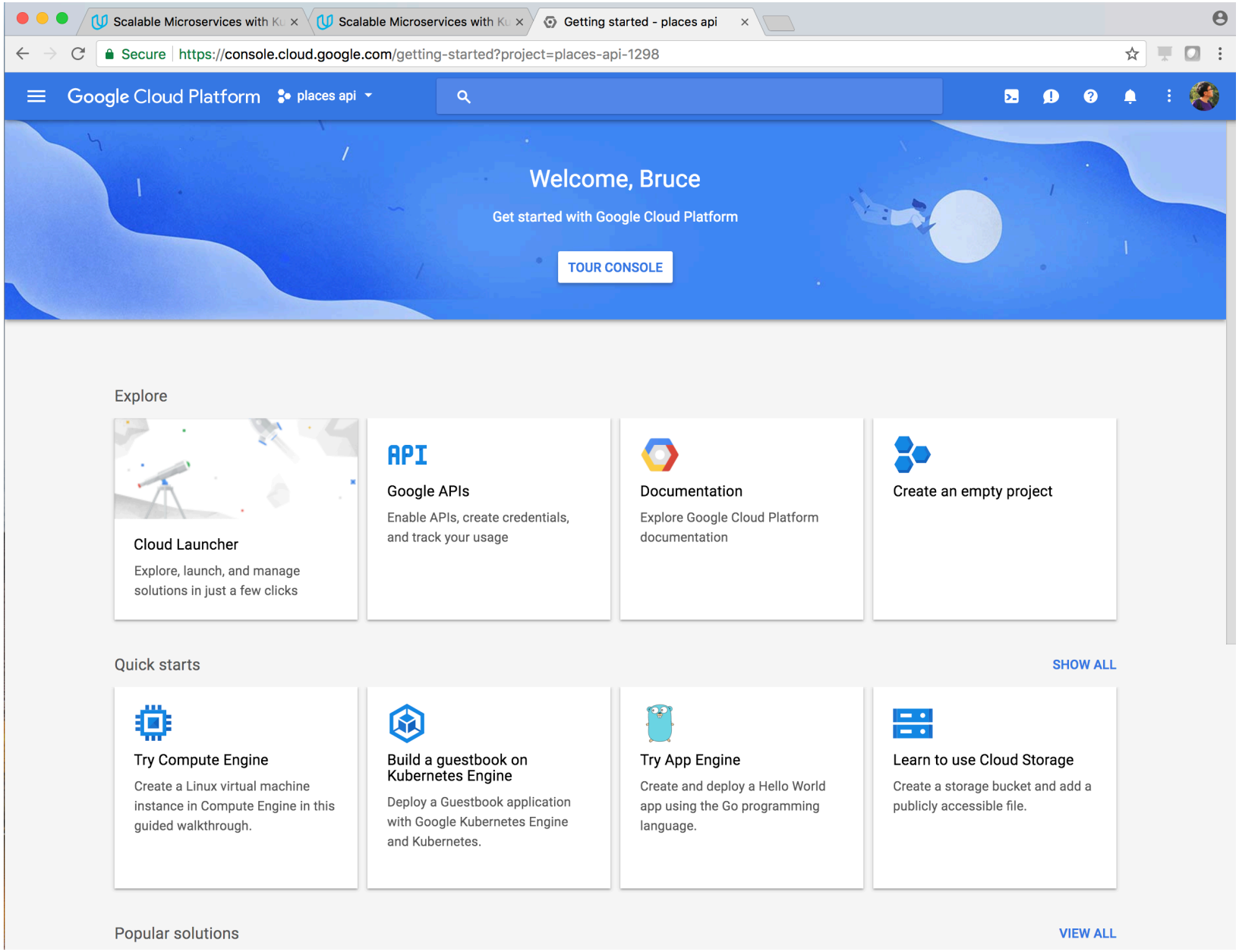
Get The Source Code

Signup Google Cloud Platform

https://console.cloud.google.com/freetrial




Secure https://console.cloud.google.com/getting-started?project=places-api-1298
Google Cloud Platform
places api
Welcome, Bruce
Get started with Google Cloud Platform
TOUR CONSOLE
Google Cloud Platform
Welcome Bruce!
Thanks for signing up for the 12-month free trial.
We've given you $300 in free trial credit to spend. If you run out of credit, don't worry, you won't be billed until you give your permission.
GOT IT
Explore
Cloud Launc
Explore, launch
solutions in jus
Platform
Quick starts
SHOW ALL
Try Compute Engine
Create a Linux virtual machine instance in Compute Engine in this guided walkthrough.
Build a guestbook on Kubernetes Engine
Deploy a Guestbook application with Google Kubernetes Engine and Kubernetes.
Try App Engine
Create and deploy a Hello World app using the Go programming language.

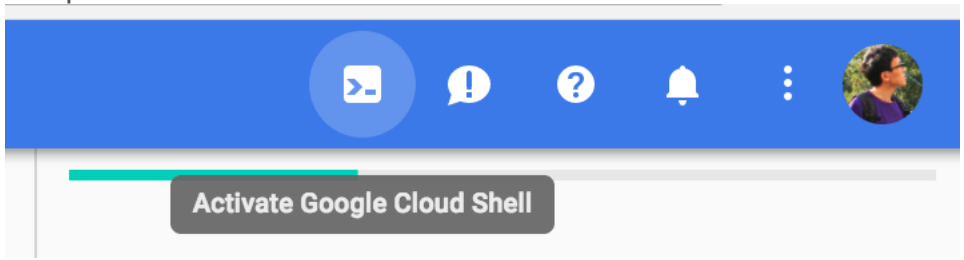

Scalable Microservices with Ku
Scalable Microservices with Ku
Getting started - places api
Secure
https://console.cloud.google.com/getting-started?project=places-api-1298
Google Cloud Platform
places api
Welcome, Bruce
Get started with Google Cloud Platform
TOUR CONSOLE
Explore
Cloud Launcher
Explore, launch, and manage solutions in just a few clicks
API
Google APIs
Enable APIs, create credentials, and track your usage
Documentation
Explore Google Cloud Platform documentation
Create an empty project
Quick starts
SHOW ALL
Try Compute Engine
Create a Linux virtual machine instance in Compute Engine in this guided walkthrough.
Build a guestbook on Kubernetes Engine
Deploy a Guestbook application with Google Kubernetes Engine and Kubernetes.
Try App Engine
Create and deploy a Hello World app using the Go programming language.
Learn to use Cloud Storage
Create a storage bucket and add a publicly accessible file.
Popular solutions
VIEW ALL

Google Cloud Platform Console

https://console.cloud.google.com

Setup GCE and Enable Cloud Shell

In this section you will create a Google Compute Engine (GCE) account. GCE will allow you to the create VMs, Networks, and Storage volumes required for this workshop. GCE also provides the **Cloud Shell**c omputing environment that will be used complete the labs.



Create a GCE Account

(As done above)

A Google Cloud Platform account is required for this course. You can use an existing GCP account or **sign up for a new one** with a valid Gmail account.

A credit card is required for Google Cloud Platform, but there is no subscription cost, you pay only for resources you use.

Create a Project

A GCP project is required for this course. You can use an existing GCP project or **create a new one**.

Your project name maybe different from your project id.

Enable Compute Engine and Container Engine APIs

In order to create the cloud resources required by this workshop, you will need to enable the following APIs using the **Google API Console**:

Compute Engine API

Container Engine API

Enable and explore Cloud Shell

Google Cloud Shell provides you with command-line access to computing resources hosted on Google Cloud Platform and is available now in the Google Cloud Platform Console. Cloud Shell makes it easy for you to **manage your Cloud Platform Console projects and resources without having to install the Google Cloud SDK and other tools on your system**. With Cloud Shell, the Cloud SDK gcloud command and other utilities you need are always available when you need them.

Explore Google Cloud Shell

Visit the **Google Cloud Shell getting started guide** and work through the exercises.

Configure Your Cloud Shell Environment

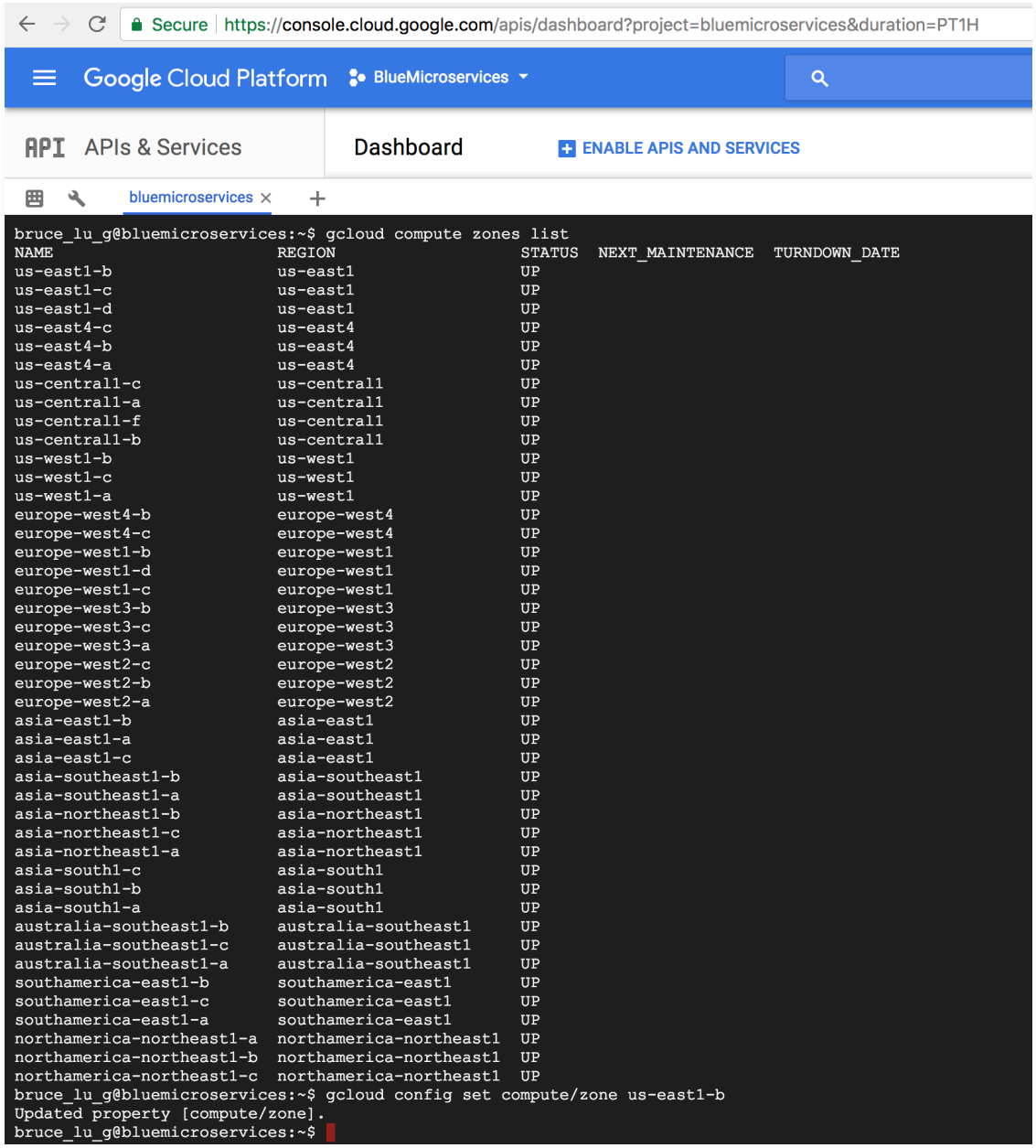
Create two Cloud Shell Sessions and run the following commands to avoid setting the compute zone.

List available time zones:

```
gcloud compute zones list
```

Set a time zone example:

```
gcloud config set compute/zone us-east1-b
```


Secure https://console.cloud.google.com/apis/dashboard?project=bluemicroservices&duration=PT1H
Google Cloud Platform
BlueMicroservices
APIs & Services
Dashboard
ENABLE APIS AND SERVICES
bluemicroservices
bruce_lu_g@bluemicroservices:~$ gcloud compute zones list
NAME REGION STATUS NEXT_MAINTENANCE TURNDOWN_DATE
us-east1-b us-east1 UP
us-east1-c us-east1 UP
us-east1-d us-east1 UP
us-east4-c us-east4 UP
us-east4-b us-east4 UP
us-east4-a us-east4 UP
us-central1-c us-central1 UP
us-central1-a us-central1 UP
us-central1-f us-central1 UP
us-central1-b us-central1 UP
us-west1-b us-west1 UP
us-west1-c us-west1 UP
us-west1-a us-west1 UP
europe-west4-b europe-west4 UP
europe-west4-c europe-west4 UP
europe-west1-b europe-west1 UP
europe-west1-d europe-west1 UP
europe-west1-c europe-west1 UP
europe-west3-b europe-west3 UP
europe-west3-c europe-west3 UP
europe-west3-a europe-west3 UP
europe-west2-c europe-west2 UP
europe-west2-b europe-west2 UP
europe-west2-a europe-west2 UP
asia-east1-b asia-east1 UP
asia-east1-a asia-east1 UP
asia-east1-c asia-east1 UP
asia-southeast1-b asia-southeast1 UP
asia-southeast1-a asia-southeast1 UP
asia-northeast1-b asia-northeast1 UP
asia-northeast1-c asia-northeast1 UP
asia-northeast1-a asia-northeast1 UP
asia-south1-c asia-south1 UP
asia-south1-b asia-south1 UP
asia-south1-a asia-south1 UP
australia-southeast1-b australia-southeast1 UP
australia-southeast1-c australia-southeast1 UP
australia-southeast1-a australia-southeast1 UP
southamerica-east1-b southamerica-east1 UP
southamerica-east1-c southamerica-east1 UP
southamerica-east1-a southamerica-east1 UP
northamerica-northeast1-a northamerica-northeast1 UP
northamerica-northeast1-b northamerica-northeast1 UP
northamerica-northeast1-c northamerica-northeast1 UP
bruce_lu_g@bluemicroservices:~$ gcloud config set compute/zone us-east1-b
Updated property [compute/zone].
bruce_lu_g@bluemicroservices:~$

```
bruce_lu_g@bluemicroservices:~$ gcloud config get-value compute/zone
Your active configuration is: [cloudshell-14635]
us-east1-b
```

Download Go:

Note: Cloud Shell comes with an installed Go, but it's not the most recent version, so you should perform the steps below to install the latest Go and set GOPATH.

```
wget https://storage.googleapis.com/golang/go1.6.2.linux-amd64.tar.gz
sudo rm -rf /usr/local/go
sudo tar -C /usr/local -xzf go1.6.2.linux-amd64.tar.gz
echo "export GOPATH=~/go" >> ~/.bashrc
source ~/.bashrc
```

Get the code:

```
mkdir -p $GOPATH/src/github.com/udacity
cd $GOPATH/src/github.com/udacity
git clone https://github.com/udacity/ud615
```

Get ready for the next lesson

```
cd ud615/app
```

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ pwd
/home/bruce_lu_g/go/src/github.com/udacity/ud615/app
```

Build and Interact with Monolith

go build -o ./bin/monolith ./monolith/

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ mkdir bin
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ go build -o ./bin/monolith ./monolith/
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ ls -l bin
total 6724
-rwxr-xr-x 1 bruce_lu_g bruce_lu_g 6884341 Mar  6 20:43 monolith
```

sudo ./bin/monolith -http 0.0.0.0:10080

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ sudo ./bin/monolith -http 0.0.0.0:10080
2018/03/07 08:21:40 Starting server...
2018/03/07 08:21:40 Health service listening on 0.0.0.0:81
2018/03/07 08:21:40 HTTP service listening on 0.0.0.0:10080

#  On another Cloud Shell terminal
bruce_lu_g@bluemicroservices:~$ curl localhost:10080
{"message":"Hello"}
bruce_lu_g@bluemicroservices:~$ curl localhost:10080/secure
authorization failed
bruce_lu_g@bluemicroservices:~$ curl localhost:10080/login -u user
Enter host password for user 'user':
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InVzZXJAZXhhbXBsZS5jb20iLCJleHAiOjE1MjA2ODg2NzAsImlhdC
I6MTUyMDQyOTQ3MCwiaXNzIjoiYXV0aC5zZXJ2aWNlIiwic3ViIjoidXNlciJ9.uEnd5M6_L09xqYTLtxjgUa3qp6lG-NQiDRhpADi9cmA"}
bruce_lu_g@bluemicroservices:~$ curl -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InVzZXJAZXhhbXBsZS5jb20iLCJleHAiOjE1MjA2ODg2NzAsImlhdCI6MTUyMDQy
OTQ3MCwiaXNzIjoiYXV0aC5zZXJ2aWNlIiwic3ViIjoidXNlciJ9.uEnd5M6_L09xqYTLtxjgUa3qp6lG-NQiDRhpADi9cmA"
localhost:10080/secure
{"message":"Hello"}

# Go back to the first terminal for more logs
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ go build -o ./bin/monolith ./monolith/
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app$ sudo ./bin/monolith -http 0.0.0.0:10080
2018/03/07 08:21:40 Starting server...
2018/03/07 08:21:40 Health service listening on 0.0.0.0:81
2018/03/07 08:21:40 HTTP service listening on 0.0.0.0:10080
127.0.0.1:48812 - - [Wed, 07 Mar 2018 08:24:16 EST] "GET / HTTP/1.1" curl/7.52.1
127.0.0.1:48823 - - [Wed, 07 Mar 2018 08:25:48 EST] "GET /secure HTTP/1.1" curl/7.52.1
127.0.0.1:48844 - - [Wed, 07 Mar 2018 08:31:10 EST] "GET /login HTTP/1.1" curl/7.52.1
127.0.0.1:48877 - - [Wed, 07 Mar 2018 08:38:36 EST] "GET /secure HTTP/1.1" curl/7.52.1
```

12 factor

Best practices for building modern software-as-a-service applications with portability, continually deployability and scalability.

More info https://12factor.net/

Quiz 12 Factor

Code is stored in Git – Codebase
Output goes to stdout – Logs
Import statements - Dependencies

Refactor to MSA

Shell 1 - build and run the hello service
go build -o ./bin/hello ./hello
sudo ./bin/hello -http 0.0.0.0:10082

Shell 2 - build and run the auth service
go build -o ./bin/auth ./auth
sudo ./bin/auth -http :10090 -health :10091

Shell 3 - interact with the auth and hello microservices
TOKEN=$(curl 127.0.0.1:10090/login -u user | jq -r '.token')
curl -H "Authorization: Bearer $TOKEN" http://127.0.0.1:10082/secure

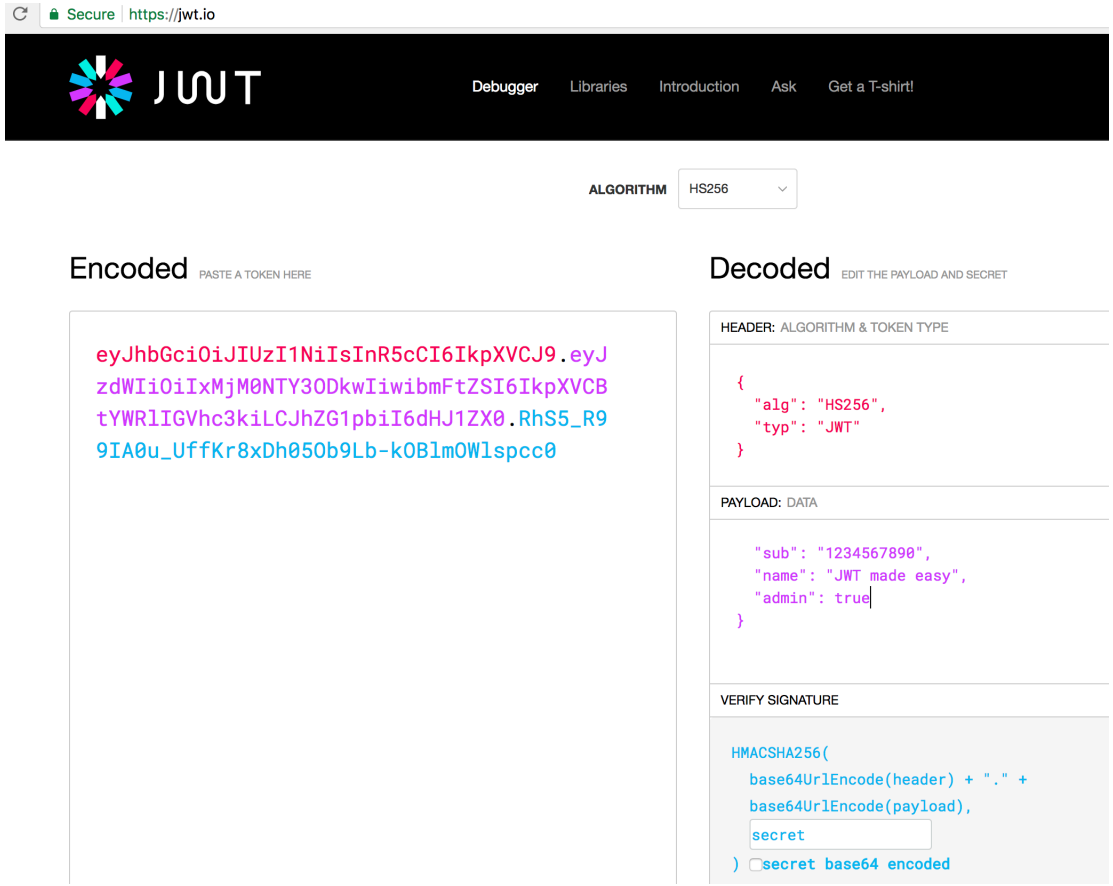
Quiz: Microservices Quiz

Microservices Architectures are solutions to which of the following problems:
[n] Slow application speed
[y] Tightly coupled components
[y] Maintenance

JWT JWS JWE

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

https://jwt.io/

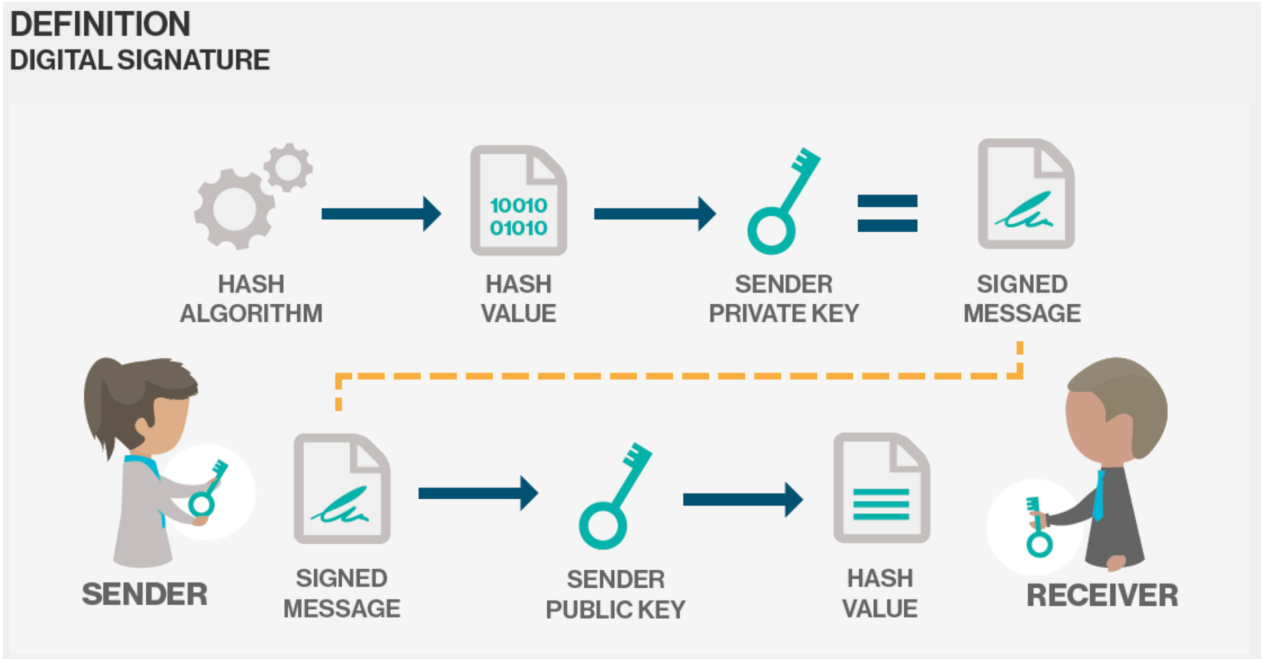


More: https://tools.ietf.org/html/rfc7519

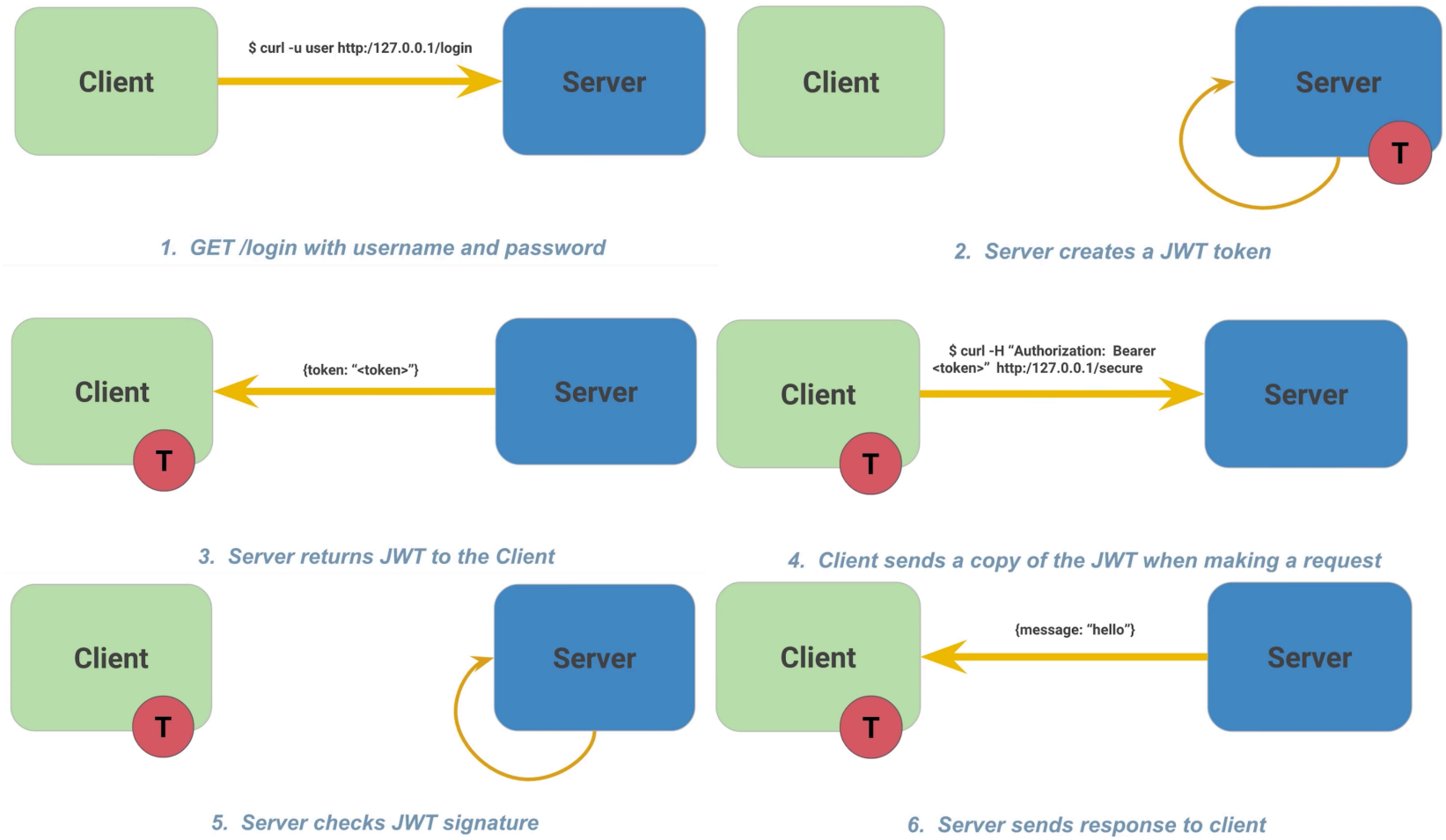
The OAuth 2.0 Authorization Framework: Bearer Token Usage

https://tools.ietf.org/html/rfc6750
http://www.rfcreader.com/#rfc6749

What is digital signature



How does JWT work



Lesson 2: Building the Containers with Docker

Why Docker

We use container technology to create(build, package), distribute and run the microservices

Docker Overview

Docker is an open source tool that makes it easy to create, distribute and run applications.

Installing apps with native OS tools

Commands from the video
Cloud shell - set compute/zone
Note - Google Cloud shell is an ephemeral instance and will reset if you don't use it for more than 30 minutes. That is why you might have to set some configuration values again

```
$ gcloud compute zones list
$ gcloud config set compute/zone us-east1-b
```

Cloud shell - launch a new VM instance

```
$ gcloud compute instances create ubuntu --image-project ubuntu-os-cloud --image ubuntu-1604-xenial-v20160420c
```

Cloud shell - log into the VM instance

```
$ gcloud compute ssh ubuntu
```

VM instance - update packages and install nginx

```
$ sudo apt-get update
$ sudo apt-get install nginx
$ nginx -v
```

VM instance - start nginx

```
$ sudo systemctl start nginx
```

Check that it's running

$ sudo systemctl status nginx
$ curl http://127.0.0.1

Quiz: Native package management

Containers Overview

Technology that makes it easy to distribute and run applications across different operation systems.

Installing Images with Docker

Commands to run (on the VM Instance)

Install Docker

```
$ sudo apt-get install docker.io
```

Check Docker images

```
$ sudo docker images
```

Pull nginx image

```
$ sudo docker pull nginx:1.10.0
$ sudo docker images
```

Verify the versions match

```
$sudo dpkg -l | grep nginx
```

If your version of nginx from native package and Docker are different, you need to update the VM instance:

```
$ sudo apt-get update
$ sudo apt-get install nginx
```

Running Images with Docker

Commands to run on the VM Instance

Run the first instance

```
$ sudo docker run -d nginx:1.10.0
```

Check if it's up

```
$ sudo docker ps
```

Run a different version of nginx

```
$ sudo docker run -d nginx:1.9.3
```

Run another version of nginx

```
$ sudo docker run -d nginx:1.10.0
```

Check how many instances are running

```
$ sudo docker ps
$ sudo ps aux | grep nginx
```

What's with the container names?
If you don't specify a name, Docker gives a container a random name, such as "stoic_williams," "sharp_bartik," "awesome_murdock," or "evil_hawking." (Stephen Hawking got no love on this one.)

These are generated from a list of adjectives and names of famous scientists and hackers. The combination of the names and adjectives is random, except for one case. Want to see what the exception is? Check it out in the Docker source code!

Talking to Docker instances

List all running container processes

```
$ sudo docker ps
```

For use in shell scripts you might want to just get a list of container IDs (-a stands for all instances, not just running, and -q is for "quiet" - show just the numeric ID):

```
$ sudo docker ps -aq
```

Inspect the container
You can use either CONTAINER ID or NAMES field, for example for a sudo docker ps output like this:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
f86cf066c304        nginx:1.10.0        "nginx -g 'daemon off"   8 minutes ago       Up 8 minutes        80/tcp, 443/tcp     sharp_bartik
```

You can use either of the following commands:

```
$ sudo docker inspect f86cf066c304
```

or

sudo docker inspect sharp_bartik

Connect to the nginx using the internal IP

Get the internal IP address either copying from the full inspect file or by assigning it to a shell variable:

```
CN="sharp_bartik"
CIP=$(sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' $CN)
curl  http://$CIP
```

You can also get all instance IDs and their corresponding IP addresses by doing this:

sudo docker inspect -f '{{.Name}} - {{.NetworkSettings.IPAddress }}' $(sudo docker ps -aq)

Stop an instance

```
$ sudo docker stop <cid>
or $ sudo docker stop $(sudo docker ps -aq)
```

Verify no more instances running

```
$ sudo docker ps
```

Remove the docker containers from the system

```
$ sudo docker rm <cid>
or $ sudo docker rm $(sudo docker ps -aq)
```

Creating your own images

Cat bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/app/hello/Dockerfile

Alpine images
Docker are using Alpine as default base of docker image. More official: https://hub.docker.com/_/alpine/

Create An Image

Commands to run on the VM Instance
Install Go
wget https://storage.googleapis.com/golang/go1.6.2.linux-amd64.tar.gz
rm -rf /usr/local/bin/go
sudo tar -C /usr/local -xzf go1.6.2.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
export GOPATH=~/go

Get the app code
mkdir -p $GOPATH/src/github.com/udacity
cd $GOPATH/src/github.com/udacity
git clone https://github.com/udacity/ud615.git

Build a static binary of the monolith app
cd ud615/app/monolith
go get -u
$ go build --tags netgo --ldflags '-extldflags "-lm -lstdc++ -static"'

Why did you have to build the binary with such an ugly command line?

You have to explicitly make the binary static. This is really important in the Docker community right now because alpine has a different implementation of libc. So your go binary wouldn't have had the lib it needed if it wasn't static. You created a static binary so that your application could be self-contained.

Create a container for the app
Look at the Dockerfile

cat Dockerfile
Build the app container

```
$ sudo docker build -t monolith:1.0.0 .
```

List the monolith image

```
$ sudo docker images monolith:1.0.0
```

Run the monolith container and get it's IP
sudo docker run -d monolith:1.0.0
sudo docker inspect <container name or cid>
or

CID=$(sudo docker run -d monolith:1.0.0)
CIP=$(sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' ${CID})
Test the container
curl <the container IP>
or

curl $CIP
Important note on security
If you are tired of typing "sudo" in front of all Docker commands, and confused why a lot of examples don't have that, please read the following article about implications on security - Why we don't let non-root users run Docker in CentOS, Fedora, or RHEL

Create Other Containers

Create docker images for the remaining microservices - auth and hello.
Repeat the steps you took for monolith.

Build the auth app
$ cd $GOPATH/src/github.com/udacity/ud615/app
cd auth
$ go build --tags netgo --ldflags '-extldflags "-lm -lstdc++ -static"'
$ sudo docker build -t auth:1.0.0 .

CID2=$(sudo docker run -d auth:1.0.0)

Build the hello app
cd $GOPATH/src/github.com/udacity/ud615/app
cd hello
$ go build --tags netgo --ldflags '-extldflags "-lm -lstdc++ -static"'
sudo docker build -t hello:1.0.0 .
CID3=$(sudo docker run -d hello:1.0.0)
See the running containers
sudo docker ps -a

Public vs Private Registries

A lot of the power of container images stems from the ability to host and download them from cloud registries. This makes it easy to share containers without having to use complex pipelines for distributing them.

When it comes to registries there are a few options. Below are a few different registries.

Docker Hub
Docker Hub is the registry we're using in this class. Go ahead and sign up for docker hub and create a repository so that you can follow along with the remaining lessons: https://hub.docker.com/

Quay
Quay is another popular registry because of it's rich automated workflow for building containers from github. https://quay.io/

Google Cloud Registry
Finally, Google Cloud Registry (GCR) is a strong options for large enterprises. https://cloud.google.com/container-registry/docs/

Comparison Of 4 Registries
A write up comparing some of the different registries: http://rancher.com/comparing-four-hosted-docker-registries/

Push Images

See all images
sudo docker images

Docker tag command help
docker tag -h

Add your own tag and push docker image to Docker Hub
$ sudo docker tag monolith:1.0.0 brucelu/monolith:1.0.0
$ sudo docker login
$ sudo docker push brucelu/monolith:1.0.0

Repeat for other two:

$ sudo docker tag auth:1.0.0 brucelu/auth:1.0.0
$ sudo docker push brucelu/auth:1.0.0
$ sudo docker tag hello:1.0.0 brucelu/ hello:1.0.0
$ sudo docker push brucelu/ hello:1.0.0

Output

```
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello               1.0.0               5e4aec669bd6        22 minutes ago      13.3 MB
auth                1.0.0               d617f9b9f01a        24 minutes ago      13.3 MB
brucelu/monolith    1.0.0               b9bcc4a933a0        32 minutes ago      13.4 MB
monolith            1.0.0               b9bcc4a933a0        32 minutes ago      13.4 MB
alpine              3.1                 2ba97bb89407        2 months ago        5.05 MB
nginx               1.10.3              0346349a1a64        11 months ago       182 MB
nginx               1.10.0              16666ff3a57f        21 months ago       183 MB
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to
https://hub.docker.com to create one.
Username: brucelu
Password:
Login Succeeded
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker push brucelu/monolith
The push refers to a repository [docker.io/brucelu/monolith]
5085231d9b2e: Pushed
33f7611be4f8: Mounted from library/alpine
1.0.0: digest: sha256:128078c8119d4d744ab7cde47ae38e67bcba2d1276c9cbe1a66d6e070b78f65f size: 739
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker tag hello:1.0.0 brucelu/hello:1.0.0
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker tag auth:1.0.0 brucelu/auth:1.0.0
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker push brucelu/hello:1.0.0
The push refers to a repository [docker.io/brucelu/hello]
244f2602c239: Pushed
33f7611be4f8: Mounted from brucelu/monolith
1.0.0: digest: sha256:0e4ca74f6f8b5ab08950ded81a9a81e2282b46ff5dca4c18b319c4843284a4e0 size: 739
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker push brucelu/auth:1.0.0
The push refers to a repository [docker.io/brucelu/auth]
5890e7edf2b5: Pushed
33f7611be4f8: Mounted from brucelu/hello
1.0.0: digest: sha256:1fec1ce355268dc2fff4d7d478ebd561ea9b40720d5e9546f41f6f85d7ddd411 size: 739
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
brucelu/hello       1.0.0               5e4aec669bd6        29 minutes ago      13.3 MB
hello               1.0.0               5e4aec669bd6        29 minutes ago      13.3 MB
auth                1.0.0               d617f9b9f01a        31 minutes ago      13.3 MB
brucelu/auth        1.0.0               d617f9b9f01a        31 minutes ago      13.3 MB
brucelu/monolith    1.0.0               b9bcc4a933a0        40 minutes ago      13.4 MB
monolith            1.0.0               b9bcc4a933a0        40 minutes ago      13.4 MB
```

```
alpine              3.1                 2ba97bb89407        2 months ago        5.05 MB
nginx               1.10.3              0346349a1a64        11 months ago       182 MB
nginx               1.10.0              16666ff3a57f        21 months ago       183 MB
bruce_lu_g@ubuntu:~/go/src/github.com/udacity/ud615/app/hello$
```

Lesson 3: Kubernetes

The infrastructure supports application at scale is as important as the application itself.

Deep Dive into Architecture

Old architecture pattern v.s. New architecture pattern = Pets v.s. Cattle

How I first learned about K8s

What is Kubernetes

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

Setting up Kubernetes for this course

Use project directory
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes

or if you are in the course repository:
cd kubernetes
Note: At any time you can clean up by running the cleanup.sh script

Provision a Kubernetes Cluster with GKE using gcloud
To complete the work in this course you going to need some tools. Kubernetes can be configured with many options and add-ons, but can be time consuming to bootstrap from the ground up. In this section you will bootstrap Kubernetes using Google Container Engine (GKE).

GKE is a hosted Kubernetes by Google. GKE clusters can be customized and supports different machine types, number of nodes, and network settings.

Use the following command to create your cluster for use for the rest of this course.

$ gcloud container clusters create k0

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ gcloud container clusters create k0
WARNING: Starting in Kubernetes v1.10, new clusters will no longer get compute-rw and storage-ro scopes added to
what is specified in --scopes (though the latter will remain included in the default --scopes). To use these
scopes, add them explicitly to --scopes. To use the new behavior, set container/new_scopes_behavior property
(gcloud config set container/new_scopes_behavior true).
Creating cluster k0...done.
Created [https://container.googleapis.com/v1/projects/bluemicroservices/zones/us-east1-b/clusters/k0].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-
east1-b/k0?project=bluemicroservices
kubeconfig entry generated for k0.
NAME  LOCATION    MASTER_VERSION  MASTER_IP      MACHINE_TYPE   NODE_VERSION  NUM_NODES  STATUS
k0    us-east1-b  1.8.7-gke.1     35.229.59.118  n1-standard-1  1.8.7-gke.1   3          RUNNING
```

Kubernetes Intro Demo

Launch a single instance:

$ kubectl run nginx --image=nginx:1.10.0

Get pods

$ kubectl get pods

Expose nginx

$ kubectl expose deployment nginx --port 80 --type LoadBalancer

List services

$ kubectl get services

Output

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl run nginx --image=nginx:1.10.0
deployment "nginx" created
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
nginx-78687579d-m9tvt   1/1       Running   0          25s
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get services
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.47.240.1   <none>        443/TCP   14m
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl expose deployment nginx --port
80 --type LoadBalancer
service "nginx" exposed
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.47.240.1     <none>        443/TCP        15m
nginx        LoadBalancer   10.47.247.198   <pending>     80:32200/TCP   5s
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
kubernetes   ClusterIP      10.47.240.1     <none>          443/TCP        16m
nginx        LoadBalancer   10.47.247.198   35.231.79.201   80:32200/TCP   1m
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$
```

Kubernetes cheat sheet

We just went over a lot and we know you're probably a little overwhelmed. Fear not! We'll be going over each of these concepts, over the next two lessons. And you can always come back to this demo -- if you need to watch it again.

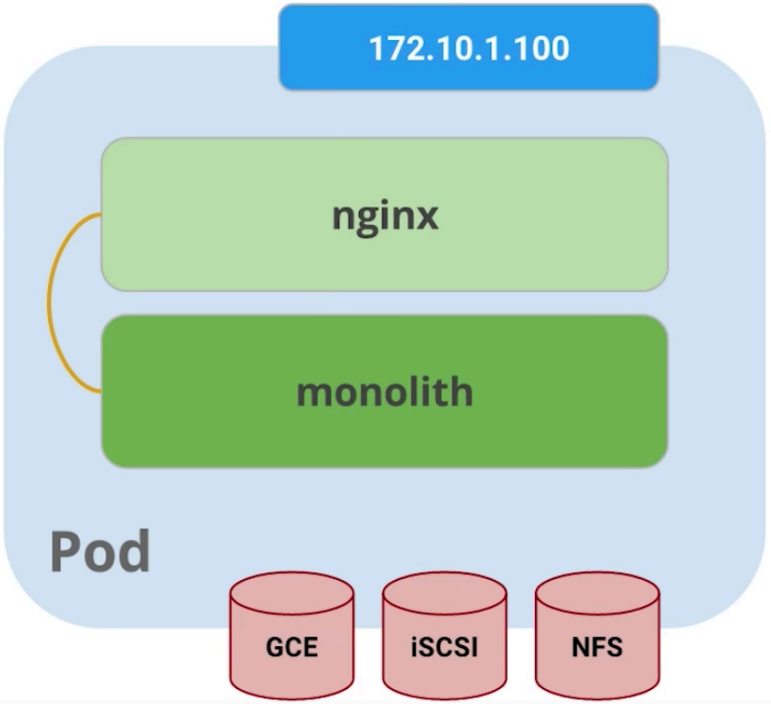
To help out, here's a Kubernetes command cheat sheet.
http://kubernetes.io/docs/user-guide/kubectl-cheatsheet/

Pods Intro

Pods

Logical Application

- One or more containers and volumes
- Shared namespaces
- One IP per pod



How does Kubernetes schedule Pods onto machines (called Nodes in Kubernetes)?

Well, Kelsey says it's a lot like tetris: **https://www.youtube.com/watch?v=Po_MEdnUVDE**
Also, it's probably a good idea to read up on Pods, my friend: **http://kubernetes.io/docs/user-guide/pods/**

Creating Pods

Explore config file

$ cat pods/monolith.yaml

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes/pods$ cat monolith.yaml
apiVersion: v1
kind: Pod
metadata:
  name: monolith
  labels:
    app: monolith
spec:
  containers:
    - name: monolith
      image: udacity/example-monolith:1.0.0
      args:
        - "-http=0.0.0.0:80"
        - "-health=0.0.0.0:81"
        - "-secret=secret"
      ports:
        - name: http
          containerPort: 80
        - name: health
          containerPort: 81
      resources:
        limits:
          cpu: 0.2
          memory: "10Mi"
```

Create the monolith pod

$ kubectl create -f pods/monolith.yaml

Examine pods

$ kubectl get pods

It may take a few seconds before the monolith pod is up and running as the monolith container image needs to be pulled from the Docker Hub before we can run it.

Use the kubectl describe command to get more information about the monolith pod.

```
$ kubectl describe pods monolith
```

Interacting with Pods

Cloud shell 1: set up port-forwarding

```
$ kubectl port-forward monolith 10080:80
```

Open new Cloud Shell session 2

```
$ curl http://127.0.0.1:10080
```

```
$ curl http://127.0.0.1:10080/secure
```

Cloud shell 2 - log in

```
$ curl -u user http://127.0.0.1:10080/login
```

```
$ curl -H "Authorization: Bearer <token>" http://127.0.0.1:10080/secure
```

View logs

```
$ kubectl logs monolith
$ kubectl logs -f monolith
```

In Cloud Shell 3

```
$ curl http://127.0.0.1:10080
```

In Cloud Shell 2

Exit log watching (Ctrl-C)

My output:

```
bruce_lu_g@bluemicroservices:~$ curl localhost:10080
{"message":"Hello"}
bruce_lu_g@bluemicroservices:~$ curl 127.0.0.1:10080/secure
authorization failed
bruce_lu_g@bluemicroservices:~$ curl -u user 127.0.0.1:10080/login
Enter host password for user 'user':
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InVzZXJAZXhhbXBsZS5jb20iLCJleHAiOjE1MjEzMDY2MzgsImlhdC
I6MTUyMTA0NzQzOCwiaXNzIjoiYXV0aC5zZXJ2aWNlIiwic3ViIjoidXNlciJ9.RyzqIJkuquzV8vhdtNDGxaCx5HrcKCX_yL56Ti6IAIA"}
bruce_lu_g@bluemicroservices:~$ curl -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InVzZXJAZXhhbXBsZS5jb20iLCJleHAiOjE1MjEzMDY2MzgsImlhdCI6MTUyMTA0
NzQzOCwiaXNzIjoiYXV0aC5zZXJ2aWNlIiwic3ViIjoidXNlciJ9.RyzqIJkuquzV8vhdtNDGxaCx5HrcKCX_yL56Ti6IAIA"
127.0.0.1:10080/secure
{"message":"Hello"}
```

You can use the kubectl exec command to run an interactive shell inside the monolith Pod. This can come in handy when you want to troubleshoot from within a container:

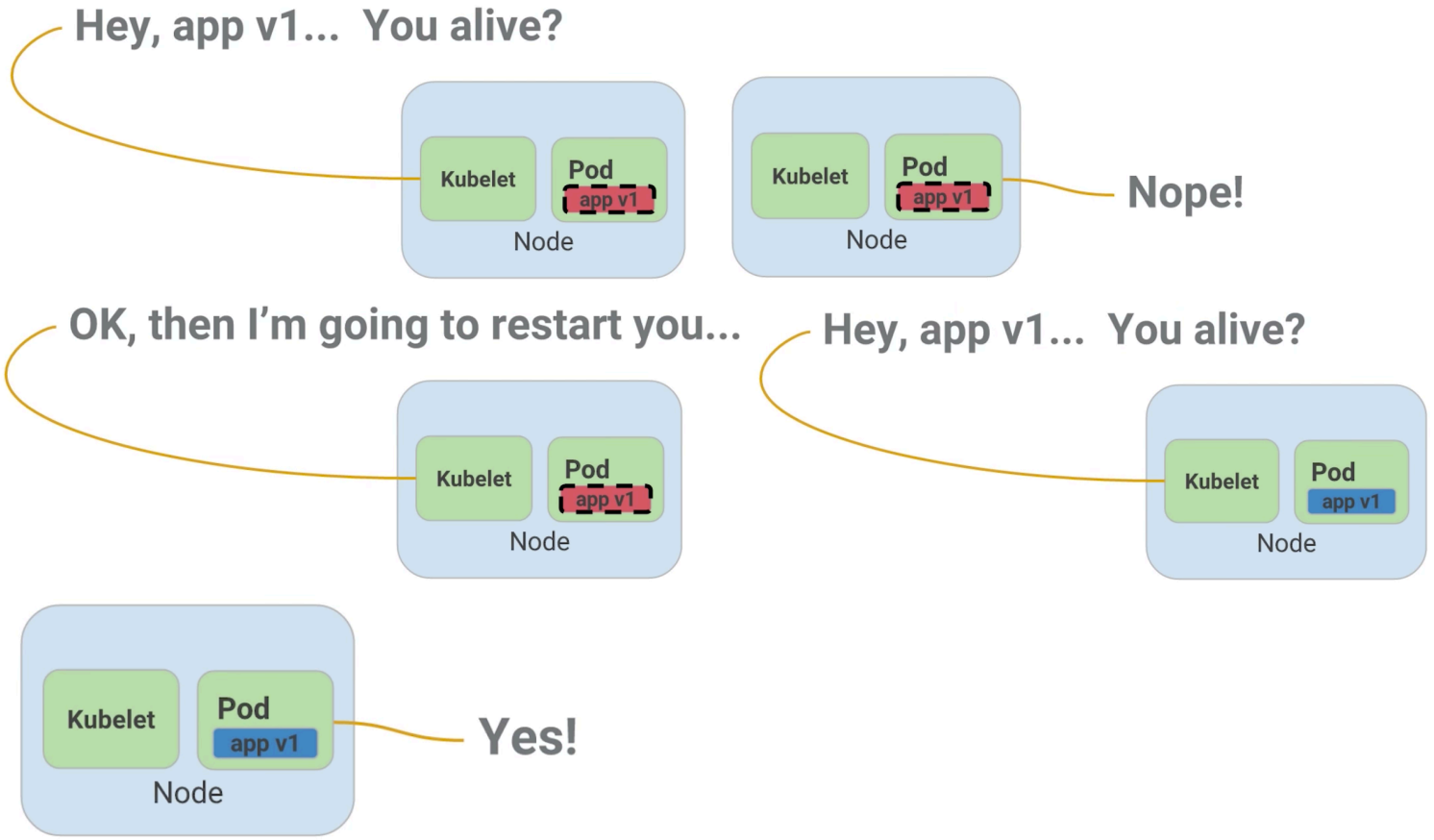
$ kubectl exec monolith --stdin --tty -c monolith /bin/sh

For example, once we have a shell into the monolith container we can test external connectivity using the ping command.

ping -c 3 google.com
When you're done with the interactive shell be sure to logout.

Exit

MHC (Monitoring and Heath Check) Overview



For the healthy-monolith pod:

How is the readiness probe performed?

How often is the readiness checked?

How often is the liveness probe checked?

bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes/pods$ cat healthy-monolith.yaml

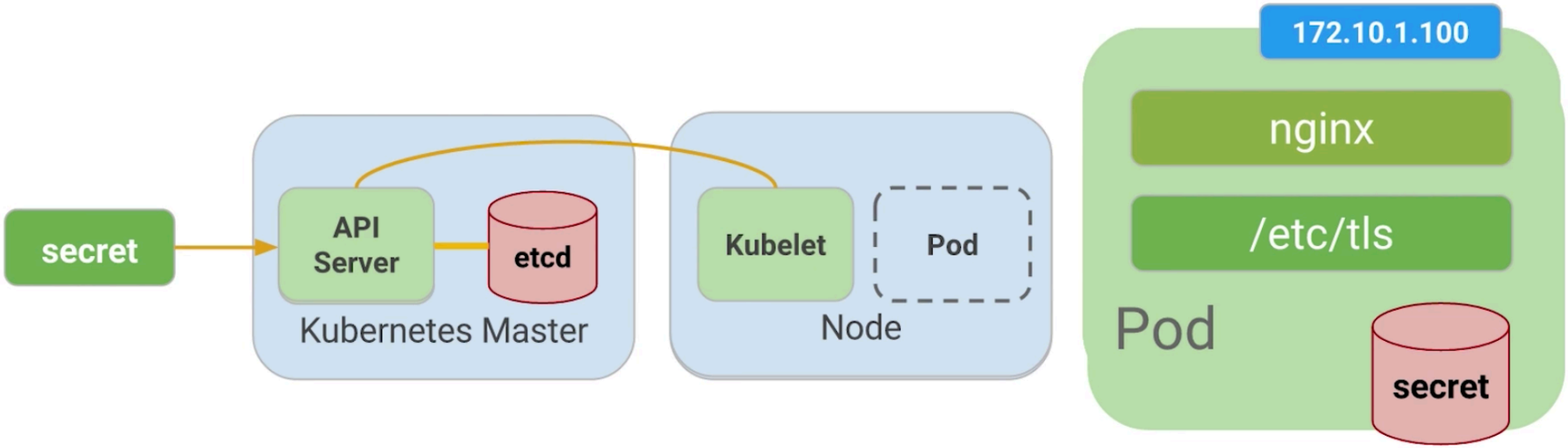
```
apiVersion: v1
kind: Pod
metadata:
  name: "healthy-monolith"
  labels:
    app: monolith
spec:
  containers:
    - name: monolith
      image: udacity/example-monolith:1.0.0
      ports:
        - name: http
          containerPort: 80
        - name: health
          containerPort: 81
      resources:
        limits:
          cpu: 0.2
          memory: "10Mi"
      livenessProbe:
        httpGet:
          path: /healthz
          port: 81
          scheme: HTTP
        initialDelaySeconds: 5
        periodSeconds: 15
        timeoutSeconds: 5
      readinessProbe:
        httpGet:
          path: /readiness
          port: 81
          scheme: HTTP
        initialDelaySeconds: 5
        timeoutSeconds: 1
```

$ kubectl describe pods healthy-monolith|grep -I readiness

More: User guide - **http://kubernetes.io/docs/user-guide/liveness/**

App Config and Security Overview

Secrets and Configmaps



etcd

etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines. It's open-source and available on GitHub. etcd gracefully handles leader elections during network partitions and will tolerate machine failure, including the leader. Your applications can read and write data into etcd.

Config docs - http://kubernetes.io/docs/user-guide/configmap/
Secrets - http://kubernetes.io/docs/user-guide/secrets/

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes/pods$ cat secure-monolith.yaml
apiVersion: v1
kind: Pod
metadata:
  name: "secure-monolith"
  labels:
    app: monolith
spec:
  containers:
    - name: nginx
      image: "nginx:1.9.14"
      lifecycle:
        preStop:
          exec:
            command: ["/usr/sbin/nginx","-s","quit"]
      volumeMounts:
        - name: "nginx-proxy-conf"
          mountPath: "/etc/nginx/conf.d"
        - name: "tls-certs"
          mountPath: "/etc/tls"
    - name: monolith
      image: "udacity/example-monolith:1.0.0"
      ports:
        - name: http
          containerPort: 80
        - name: health
          containerPort: 81
      resources:
        limits:
          cpu: 0.2
          memory: "10Mi"
      livenessProbe:
        httpGet:
          path: /healthz
          port: 81
          scheme: HTTP
        initialDelaySeconds: 5
        periodSeconds: 15
        timeoutSeconds: 5
      readinessProbe:
        httpGet:
```

```
          path: /readiness
          port: 81
          scheme: HTTP
        initialDelaySeconds: 5
        timeoutSeconds: 1
  volumes:
    - name: "tls-certs"
      secret:
        secretName: "tls-certs"
    - name: "nginx-proxy-conf"
      configMap:
        name: "nginx-proxy-conf"
        items:
          - key: "proxy.conf"
            path: "proxy.conf"
```

What is yaml and how to parse it

http://www.ruanyifeng.com/blog/2016/07/yaml.html

http://nodeca.github.io/js-yaml/

How to generate/update tls cert

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes/tls$ cat update-tls.sh
#!/bin/sh
set -e

openssl genrsa \
  -out ca-key.pem 2048

openssl req \
  -x509 \
  -new \
  -nodes \
  -key ca-key.pem \
  -days 3560 \
  -extensions v3_ca \
  -out ca.pem \
  -subj "/O=Kubernetes"

openssl x509 -in ca.pem -text
```

```
openssl genrsa \
  -out key.pem 2048

openssl req \
  -new \
  -key key.pem \
  -out csr.pem \
  -subj "/O=Kubernetes/CN=*.example.com"

openssl x509 \
  -req \
  -in csr.pem \
  -CA ca.pem \
  -CAkey ca-key.pem \
  -CAcreateserial \
  -out cert.pem \
  -extensions v3_ca \
  -extfile ssl-extensions-x509.cnf \
  -days 3560

openssl x509 -in cert.pem -text

rm ca.srl
```

More about self-signed CA & certs

https://segmentfault.com/a/1190000002569859

Creating Secrets

Commands from video
cd ~/go/src/github.com/udacity/ud615/kubernetes
ls tls
The cert.pem and key.pem files will be used to secure traffic on the monolith server and the ca.pem will be used by HTTP clients as the CA to trust. Since the certs being used by the monolith server where signed by the CA represented by ca.pem, HTTP clients that trust ca.pem will be able to validate the SSL connection to the monolith server.

Use kubectl
to create the tls-certs secret from the TLS certificates stored under the tls directory:

$ kubectl create secret generic tls-certs --from-file=tls/

kubectl will create a key for each file in the tls directory under the tls-certs secret bucket. Use the kubectl describe command to verify that:

$ kubectl get secrets

$ kubectl describe secrets tls-certs

Next we need to create a configmap entry for the proxy.conf nginx configuration file using the kubectl create configmap command:

$ kubectl create configmap nginx-proxy-conf --from-file=nginx/proxy.conf

Use the kubectl describe configmap command to get more details about the nginx-proxy-conf configmap entry:

$ kubectl describe configmap nginx-proxy-conf

TLS and SSL
TLS and SSL can be confusing topics. Here's a good primer for understanding the basics:
https://en.wikipedia.org/wiki/Transport_Layer_Security

My output

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl create secret generic tls-
certs --from-file=tls/
secret "tls-certs" created
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get secrets
NAME                  TYPE                                  DATA      AGE
default-token-fbwks   kubernetes.io/service-account-token   3         1d
tls-certs             Opaque                                6         14s
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl describe secrets tls-certs
Name:         tls-certs
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque
```

```
Data
====
ca-key.pem:               1675 bytes
ca.pem:                   1099 bytes
cert.pem:                 1253 bytes
key.pem:                  1679 bytes
ssl-extensions-x509.cnf:  275 bytes
update-tls.sh:            610 bytes
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl create configmap nginx-proxy-
conf --from-file=nginx/proxy.conf
configmap "nginx-proxy-conf" created
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl describe configmap nginx-
proxy-conf
Name:         nginx-proxy-conf
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
proxy.conf:
----
server {
  listen 443;
  ssl    on;

  ssl_certificate     /etc/tls/cert.pem;
  ssl_certificate_key /etc/tls/key.pem;

  location / {
    proxy_pass http://127.0.0.1:80;
  }
}

Events:  <none>
```

Access a Secure HTTPS Endpoint

Commands from video

```
$ cat pods/secure-monolith.yaml
```

Create the secure-monolith Pod using kubectl.

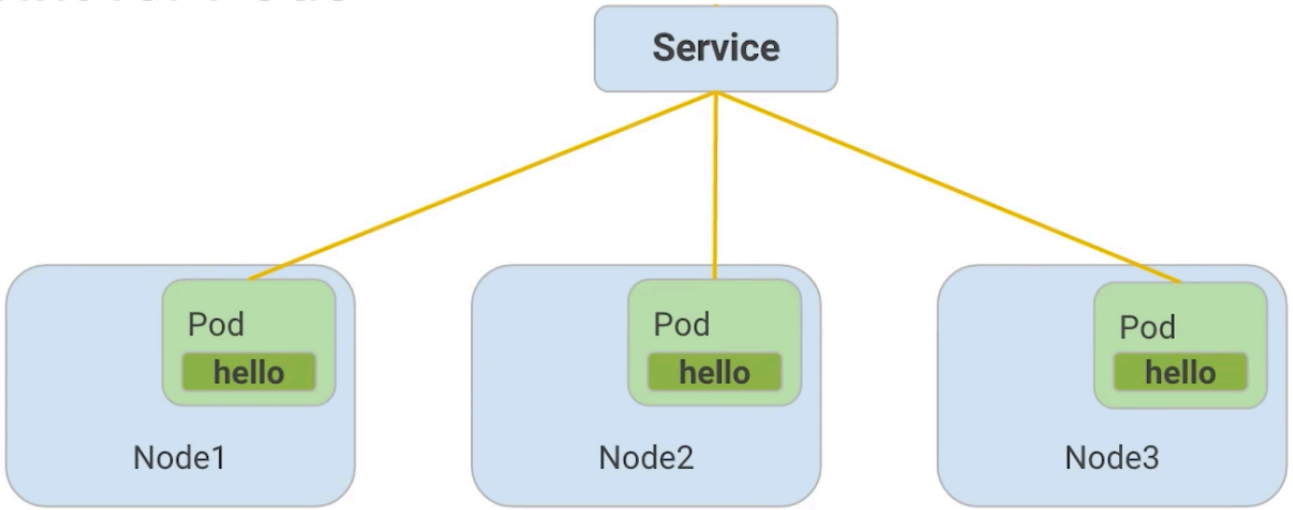
```
$ kubectl create -f pods/secure-monolith.yaml
$ kubectl get pods secure-monolith

$ kubectl port-forward secure-monolith 10443:443

$ curl --cacert tls/ca.pem https://127.0.0.1:10443

$ kubectl logs -c nginx secure-monolith
# -c nginx specifies the container name
```

Services Overview

Services

Persistent Endpoint for Pods

- Use Labels to Select Pods
- Internal or External IPs



Services overview - http://kubernetes.io/docs/user-guide/services/

Creating a Service

$ cd ~/go/src/github.com/udacity/ud615/kubernetes

$ cat services/monolith.yaml

$ gcloud compute firewall-rules create allow-monolith-nodeport --allow=tcp:31000

Added firewall rule to allow the traffic coming in

Adding Labels to Pods

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ gcloud compute instances list
NAME                                ZONE        MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP     STATUS
gke-k0-default-pool-a0d050ab-cb6f  us-east1-b  n1-standard-1               10.142.0.3   35.229.50.7     RUNNING
gke-k0-default-pool-a0d050ab-jbf1  us-east1-b  n1-standard-1               10.142.0.4   35.227.27.160   RUNNING
gke-k0-default-pool-a0d050ab-kjl8  us-east1-b  n1-standard-1               10.142.0.5   35.227.42.219   RUNNING
ubuntu                              us-east1-b  n1-standard-1               10.142.0.2   35.196.97.92    RUNNING
```

$ kubectl get pods -l "app=monolith"

$ kubectl get pods -l "secure=enabled"

$ kubectl label pods secure-monolith "secure=enabled"

$ kubectl get pods -l "secure=enabled"

$ kubectl describe pods secure-monolith|grep -A 2 Label

$ curl -k https://35.227.27.160:31000

Lesson 4: Deploying Microservices

Intro

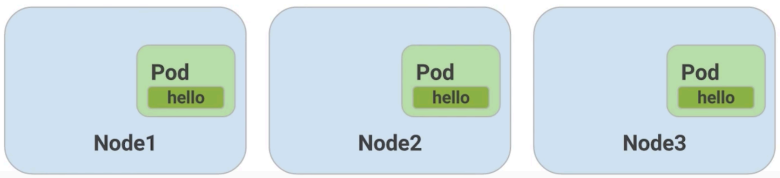
Desired state. You tell Kube the state and it handles it for you.

Deployment Overview

Deployments

Drive current state towards desired state

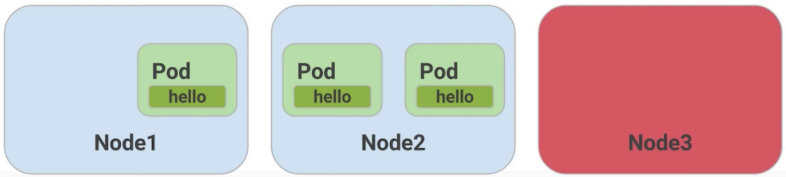
app: hello
replicas: 3



Deployments

Drive current state towards desired state

app: hello
replicas: 3



Creating Deployments

```
cd ~/go/src/github.com/udacity/ud615/kubernetes
```

$ cat deployments/auth.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: auth
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: auth
        track: stable
    spec:
      containers:
        - name: auth
          image: "udacity/example-auth:1.0.0"
          ports:
            - name: http
```

```
        containerPort: 80
      - name: health
        containerPort: 81
    resources:
      limits:
        cpu: 0.2
        memory: "10Mi"
    livenessProbe:
      httpGet:
        path: /healthz
        port: 81
        scheme: HTTP
      initialDelaySeconds: 5
      periodSeconds: 15
      timeoutSeconds: 5
    readinessProbe:
      httpGet:
        path: /readiness
        port: 81
        scheme: HTTP
      initialDelaySeconds: 5
      timeoutSeconds: 1
```

$ kubectl create -f deployments/auth.yaml

```
deployment "auth" created
```

$ kubectl describe deployments auth

```
Name:                   auth
Namespace:              default
CreationTimestamp:      Wed, 14 Mar 2018 22:02:42 -0400
Labels:                 app=auth
                        track=stable
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=auth,track=stable
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
```

```
Pod Template:
  Labels:  app=auth
           track=stable
  Containers:
   auth:
    Image:  udacity/example-auth:1.0.0
    Ports:  80/TCP, 81/TCP
    Limits:
      cpu:        200m
      memory:     10Mi
    Liveness:     http-get http://:81/healthz delay=5s timeout=5s period=15s #success=1 #failure=3
    Readiness:    http-get http://:81/readiness delay=5s timeout=1s period=10s #success=1 #failure=3
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   auth-5f7cd8c69d (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  39s   deployment-controller  Scaled up replica set auth-5f7cd8c69d to 1
```

$ kubectl create -f services/auth.yaml

```
service "auth" created
```

$ kubectl create -f deployments/hello.yaml

```
deployment "hello" created
```

$ kubectl create -f services/hello.yaml

```
service "hello" created
```

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl create configmap nginx-
frontend-conf --from-file=nginx/frontend.conf
configmap "nginx-frontend-conf" created
```

```
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl create -f
deployments/frontend.yaml
deployment "frontend" created
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl create -f
services/frontend.yaml
service "frontend" created
# kubectl delete configmap nginx-frontend-conf for deletion
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get services frontend
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
frontend   LoadBalancer   10.47.248.127   <pending>     443:31281/TCP   22s
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ kubectl get services frontend
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)         AGE
frontend   LoadBalancer   10.47.248.127   35.196.195.114   443:31281/TCP   1m
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$ curl -k https://35.196.195.114
{"message":"Hello"}
```

More yaml (https://github.com/udacity/ud615/tree/master/kubernetes)

$ cat services/auth.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: "auth"
spec:
  selector:
    app: "auth"
  ports:
    - protocol: "TCP"
      port: 80
      targetPort: 80
```

$ cat deployments/hello.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 1
  template:
    metadata:
```

```
      labels:
        app: hello
        track: stable
    spec:
      containers:
        - name: hello
          image: "udacity/example-hello:1.0.0"
          ports:
            - name: http
              containerPort: 80
            - name: health
              containerPort: 81
          resources:
            limits:
              cpu: 0.2
              memory: "10Mi"
          livenessProbe:
            httpGet:
              path: /healthz
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            periodSeconds: 15
            timeoutSeconds: 5
          readinessProbe:
            httpGet:
              path: /readiness
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            timeoutSeconds: 1
```

$ cat services/hello.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: "hello"
spec:
  selector:
    app: "hello"
```

```
  ports:
    - protocol: "TCP"
      port: 80
      targetPort: 80
```

$ cat deployments/frontend.yaml

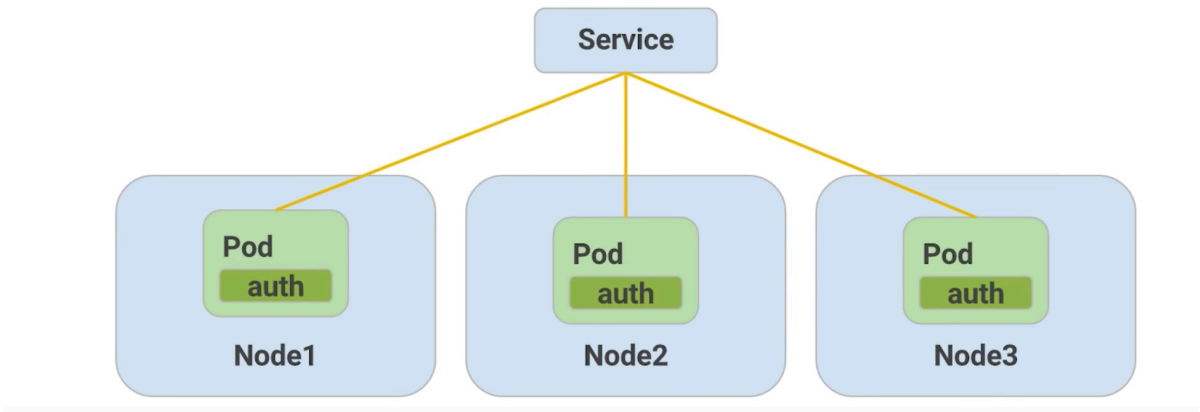
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: frontend
        track: stable
    spec:
      containers:
        - name: nginx
          image: "nginx:1.9.14"
          lifecycle:
            preStop:
              exec:
                command: ["/usr/sbin/nginx","-s","quit"]
          volumeMounts:
            - name: "nginx-frontend-conf"
              mountPath: "/etc/nginx/conf.d"
            - name: "tls-certs"
              mountPath: "/etc/tls"
      volumes:
        - name: "tls-certs"
          secret:
            secretName: "tls-certs"
        - name: "nginx-frontend-conf"
          configMap:
            name: "nginx-frontend-conf"
            items:
              - key: "frontend.conf"
                path: "frontend.conf"
```

$ cat services/frontend.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: "frontend"
spec:
  selector:
    app: "frontend"
  ports:
    - protocol: "TCP"
      port: 443
      targetPort: 443
  type: LoadBalancer
```

Scaling Overview

Kubernetes handles the scaling automatically according to the deployment replica



Scaling Deployments

$ kubectl get replicasets

```
NAME                  DESIRED   CURRENT   READY     AGE
auth-5f7cd8c69d       1         1         1         35m
frontend-96d5c4674    1         1         1         22m
hello-54564f9fd       1         1         1         27m
```

$ vim deployments/hello.yaml

```
# set replicas: 3
```

$ kubectl apply deployments/hello.yaml

$ kubectl get replicasets

```
NAME                  DESIRED   CURRENT   READY     AGE
auth-5f7cd8c69d       1         1         1         56m
frontend-96d5c4674    1         1         1         43m
hello-54564f9fd       3         3         3         48m
```

$ kubectl get pods

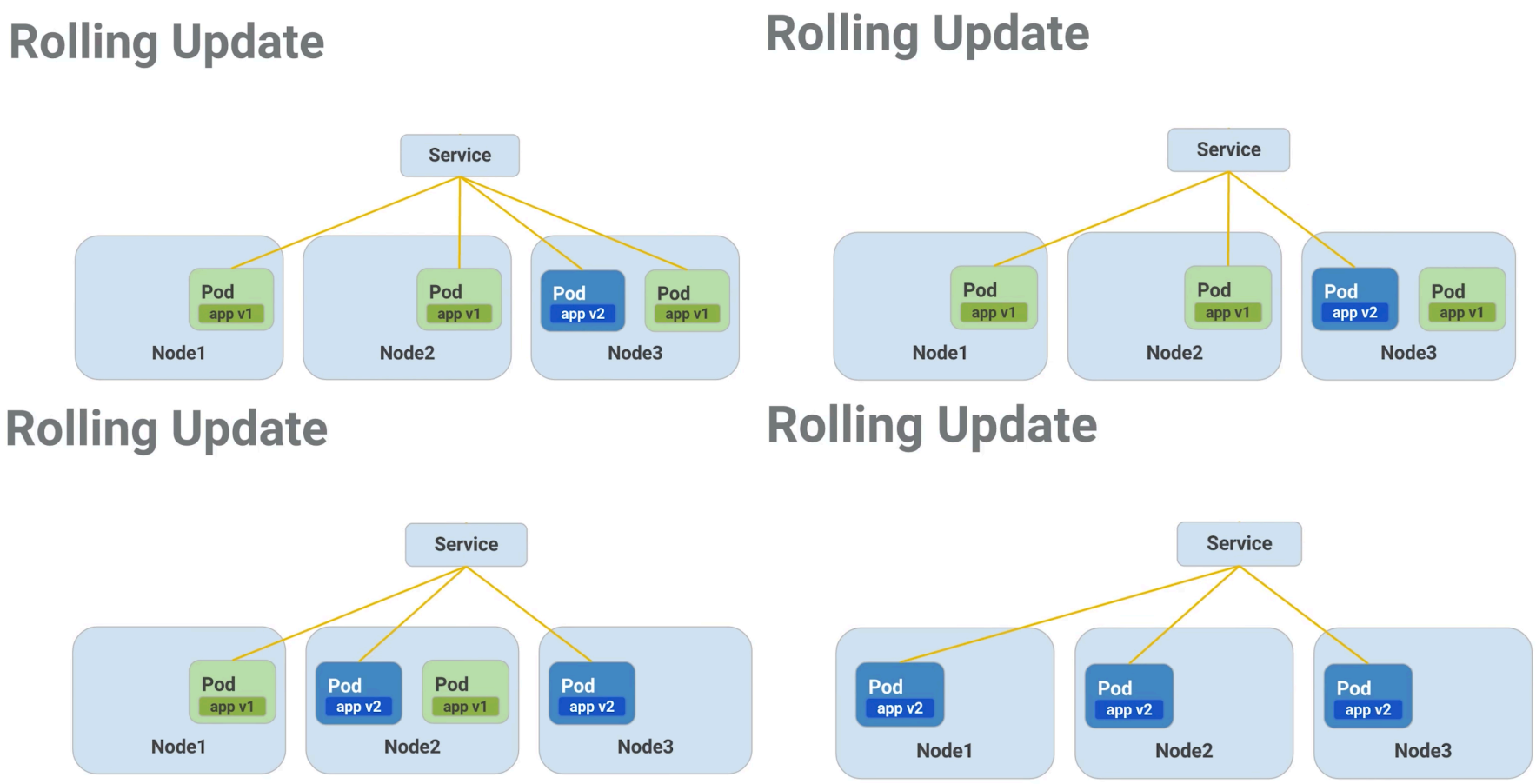
```
NAME                        READY     STATUS    RESTARTS   AGE
auth-5f7cd8c69d-6zht7       1/1       Running   0          56m
frontend-96d5c4674-rhscw    1/1       Running   0          43m
hello-54564f9fd-6wtz9       1/1       Running   0          48m
hello-54564f9fd-6x8dq       1/1       Running   0          11m
hello-54564f9fd-jzk96       1/1       Running   0          11m
```

$ kubectl get services

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
auth         ClusterIP      10.47.247.33    <none>           80/TCP           50m
frontend     LoadBalancer   10.47.248.127   35.196.195.114   443:31281/TCP    44m
hello        ClusterIP      10.47.242.184   <none>           80/TCP           49m
kubernetes   ClusterIP      10.47.240.1     <none>           443/TCP          2d
monolith     NodePort       10.47.249.70    <none>           443:31000/TCP    4h
```

```
$ curl -k https://35.196.195.114
{"message":"Hello"}
```

Updating Overview



Rolling Updates

$ cp deployments/auth.yaml deployments/auth.yaml.bk

$ vim deployments/auth.yaml

Modify version from 1.0.0 to 2.0.0

image: "udacity/example-auth:2.0.0"

$ kubectl apply -f deployments/auth.yaml

```
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment "auth" configured
```

$ kubectl describe deployments auth

```
Name:                   auth
Namespace:              default
CreationTimestamp:      Wed, 14 Mar 2018 22:02:42 -0400
Labels:                 app=auth
                        track=stable
Annotations:            deployment.kubernetes.io/revision=2
                        kubectl.kubernetes.io/last-applied-
configuration={"apiVersion":"extensions/v1beta1","kind":"Deployment","metadata":{"annotations":{},"name":"auth","
namespace":"default"},"spec":{"replicas":1,"template...
Selector:               app=auth,track=stable
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  app=auth
           track=stable
  Containers:
   auth:
    Image:  udacity/example-auth:2.0.0
    Ports:  80/TCP, 81/TCP
    Limits:
      cpu:        200m
      memory:     10Mi
    Liveness:     http-get http://:81/healthz delay=5s timeout=5s period=15s #success=1 #failure=3
    Readiness:    http-get http://:81/readiness delay=5s timeout=1s period=10s #success=1 #failure=3
```

```
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   auth-7cbb5b46bc (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  1m    deployment-controller  Scaled up replica set auth-7cbb5b46bc to 1
  Normal  ScalingReplicaSet  1m    deployment-controller  Scaled down replica set auth-5f7cd8c69d to 0
```

$ kubectl get pods

```
NAME                        READY     STATUS    RESTARTS   AGE
auth-7cbb5b46bc-gqx7w       1/1       Running   0          2m
frontend-96d5c4674-rhscw    1/1       Running   0          1h
hello-54564f9fd-6wtz9       1/1       Running   0          1h
hello-54564f9fd-6x8dq       1/1       Running   0          32m
hello-54564f9fd-jzk96       1/1       Running   0          32m
```

$ kubectl describe pods auth-7cbb5b46bc-gqx7w

```
Name:           auth-7cbb5b46bc-gqx7w
Namespace:      default
Node:           gke-k0-default-pool-a0d050ab-kjl8/10.142.0.5
Start Time:     Wed, 14 Mar 2018 23:17:23 -0400
Labels:         app=auth
                pod-template-hash=3766160267
                track=stable
Annotations:    kubernetes.io/created-
by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":"ReplicaSet","namespace":"default","name":
"auth-7cbb5b46bc","uid":"6153c2de-27ff-11e8-9176-42010a8e00bc","a...
Status:         Running
IP:             10.44.1.9
Created By:     ReplicaSet/auth-7cbb5b46bc
Controlled By:  ReplicaSet/auth-7cbb5b46bc
Containers:
  auth:
```

```
    Container ID:   docker://e0bedf549f2e69414112750853efde41035f61764d933e4c6f02d5da6bc21c03
    Image:          udacity/example-auth:2.0.0
    Image ID:       docker-pullable://udacity/example-
auth@sha256:02e142517709d35ca6cd51f8c1416a1c9514bcc0c369126fa0942007d23cf79a
    Ports:          80/TCP, 81/TCP
    State:          Running
      Started:      Wed, 14 Mar 2018 23:17:26 -0400
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     200m
      memory:  10Mi
    Requests:
      cpu:        200m
      memory:     10Mi
    Liveness:     http-get http://:81/healthz delay=5s timeout=5s period=15s #success=1 #failure=3
    Readiness:    http-get http://:81/readiness delay=5s timeout=1s period=10s #success=1 #failure=3
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-fbwks (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-fbwks:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-fbwks
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  <none>
Tolerations:     node.alpha.kubernetes.io/notReady:NoExecute for 300s
                 node.alpha.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From                                          Message
  ----    ------                 ----  ----                                          -------
  Normal  Scheduled              3m    default-scheduler                             Successfully assigned auth-
7cbb5b46bc-gqx7w to gke-k0-default-pool-a0d050ab-kjl8
  Normal  SuccessfulMountVolume  3m    kubelet, gke-k0-default-pool-a0d050ab-kjl8  MountVolume.SetUp succeeded
for volume "default-token-fbwks"
```

```
  Normal  Pulling                 3m    kubelet, gke-k0-default-pool-a0d050ab-kj18  pulling image
"udacity/example-auth:2.0.0"
  Normal  Pulled                  3m    kubelet, gke-k0-default-pool-a0d050ab-kj18  Successfully pulled image
"udacity/example-auth:2.0.0"
  Normal  Created                 3m    kubelet, gke-k0-default-pool-a0d050ab-kj18  Created container
  Normal  Started                 3m    kubelet, gke-k0-default-pool-a0d050ab-kj18  Started container
bruce_lu_g@bluemicroservices:~/go/src/github.com/udacity/ud615/kubernetes$
```

Thanks to microservices top expert Adrian, Google's Kelsey (right) and Easter (middle)