Infrastructure as Code
So far..

- **Platform-as-a-Service**
  - Write a program, deploy it on auto-scaling platform
  - Infrastructure completely abstracted out
  - No management costs, but not flexible
    - Coarse control over how and when to scale infrastructure
    - Fixed platform environment
- **Infrastructure-as-a-Service**
  - Low-level infrastructure exposed
  - Manage dozens of custom containers and VMs needed to run your site
  - Flexible, but high management costs
    - Manual creation and deployment of nodes time-consuming
    - Networking containers and load balancing complex/error-prone
Infrastructure as Code

- Obtain the automation and low management costs of PaaS while allowing low-level access to resources as in IaaS
  - Use code to configure, deploy, and manage VMs and containers
- Recall SDNs
  - Programmatic configuration of routes, virtual networks, traffic engineering policies
  - IAC is its analogy for virtual machines and containers
Infrastructure as Code

- Two pieces
  - Configuration management (setting up images)
  - Deployment management (run-time orchestration-monitoring of instances)
Virtual machines

- Configuration management
  - Imperative approaches (run commands on VMs all at once)
    - Vagrant
    - Ansible (playbooks)
    - Chef (recipes)
  - Declarative approaches (Puppet)
  - Both (SaltStack)
- Deployment management (orchestration)
  - Create, deploy, and monitor VMs over multiple cloud providers (Terraform)
Containers

- Configuration management
  - Setup and configure collections of containers (Docker)
- Deployment management (orchestration)
  - Automatically configure, deploy, and monitor clusters of containers, as well as keep them running
- This class…
Container orchestration

- Two pieces
  - A declarative, programmatic way of specifying containers and their run-time configuration
  - A daemon (orchestration controller) that monitors and maintains specification at run-time
Container orchestration tasks

- Scaling and replication (run N copies or autoscale based on metric)
  - e.g. “10 versions of a container should run at all times” or "keep adding containers until load < 0.5"
- Allocating and scheduling of containers onto machines by resource needs or affinity requirements
- Configuring network connectivity and load balancing (rather than having to specify IP addresses)
- Health monitoring and handling machine failures
- Automated software updates
Example: Google Borg (2003)

- Ops tool to automate cluster management
  - https://research.google.com/pubs/pub43438.html
- Declarative language specifies what to run and how to run it
- Borg controller monitors and maintains specification when nodes go down
- Allows developer to focus on application logic, not machine management
- Obviates the need to ssh into ANY machines
- Re-implemented spun out as an open-source project called…
Kubernetes
Kubernetes (2014)

- Open-source reimplementation of Google’s “Borg” in Go
- White-board level specification of computing infrastructure
  - “Build-your-own” PaaS via specification
  - Logical system view decoupled from cloud provider (e.g. not tied to gcloud, aws)
- Manages all Google services (Mail, Search, Maps etc)
- Portable
  - Can use for on premises, cloud, or multi-cloud deployments (configurations run the same)
  - In DevOps environments allows one to move app between development and operations seamlessly
"The cloud now is way too focused about the infrastructure, the machines, the disks, and that is not the path to productivity... You don't really want to know the machines exist. It would be nice if we could pretend they don't exist at all and Kubernetes is absolutely a step in that direction"

"The new cloud is about services and APIs and has nothing to do with infrastructure. (The infrastructure) is there, (Google) will manage it, we will make it secure, but as a developer, you shouldn't need to know."

Eric Brewer (4/24/2018)
Example

- Run 10,000 copies of a site inside a data center...
- Painful!
job hello_world = {
    runtime = { cell = 'ic' } // Cell (cluster) to run in
    binary = './hello_world_webserver' // Program to run
    args = { port = '%port%' } // Command line parameters
    requirements = { // Resource requirements (optional)
        ram = 100M
        disk = 100M
        cpu = 0.1
    }
    replicas = 10000 // Number of tasks
}
Kubernetes concepts

- **Nodes** = machine running container
  - e.g. a Compute Engine instance
- **Pods**
  - One or more tightly coupled containers sharing storage, configuration on a node
  - Web front end and its logging facility run on separate containers, but in a single pod
- **Services**
  - Defines a way to access pods externally via an IP address and port
- **Controller**
  - Service that monitors pods to ensure the desired number of replicas are always running
Kubernetes

• The Illustrated Children’s guide (8:20)
  https://www.youtube.com/watch?v=4ht22ReBjno

• Kubecraft (1:50)
  https://www.youtube.com/watch?v=A4qwsSE1dHE
Deploying Kubernetes

- **kubectl** command to interact with master controller

  - `kubectl create -f demo_pod.yaml`

apiVersion: v1
kind: Pod
metadata:
  name: test
  labels:
    app: test
spec:
  containers:
  - name: backend
    image: us.gcr.io/cc-p22/task2-container1:v0
    ports:
    - name: http
      containerPort: 8080
  ...

Node 1

Pod 1

- task2-container1
- task2-container2

Node 2

Pod 2

- task2-container1
- task2-container2
Case study: Pokémon Go

- Published by Niantic
  - Spin-out of Google Earth's toy app Ingress
    - Single consistent world state across all users
    - Has an $n^2$ scaling problem
- Built on Google Cloud Platform
  - Initially on App Engine
  - Provisioned for launch numbers based on prior launches of other top-tier mobile games multiplied by 5
- Released July 5, 2016
Case study: Pokemon Go

- But then,

- Provisioned number for global use met in just two countries
- Within 1 week, 50x more users than expected
  - 8.7 billion km walked in first 6 months (past Pluto or 200k times around Earth)
Case study: Pokemon Go

- Scaled seamlessly* with no change to software
- Built on Java (server)
- C#, C++, Objective C, Unity (client)
- Cloud Datastore (Distributed NoSQL)
- Shifted to Kubernetes for better control over scaling components
Case study: Home Depot

- Single monolithic site with single monolithic database
- Container migration for developer efficiency
  - Site split into microservices
    - Store finder service
    - Promotions service
    - Product information service
  - Individual DevOps teams develop individual microservice
  - Removes centralized bottleneck in managing site
- Microservices resources
Over the last year, GitHub has gradually evolved the infrastructure that runs the Ruby on Rails application responsible for github.com and api.github.com. We reached a big milestone recently: all web and API requests are served by containers running in Kubernetes clusters deployed on our metal cloud. Moving a critical application to Kubernetes was a fun.

CCP Games: Unleashing developer creativity with Kubernetes

CCP Games, based in Reykjavik, Iceland has been a leading game developer since 1997, authoring ground-breaking games including the critically acclaimed science-fiction game EVE Online. As its games have
Kubernetes Engine

AWS Elastic Kubernetes Service, AWS Fargate
Azure Kubernetes Service
Google Kubernetes Engine (GKE)

- Hosted Kubernetes for running controller on GCP
- Similar to AWS EC2 Container Service, Azure Container Service
Other container services

- Container Registry (gcr.io)
  - Store container images used in lab within GCP for quick instantiation
  - Akin to having DockerHub on GCP, but containers private to project
- Container Builder
  - Remotely build container images on GCP
Beyond Kubernetes

- Deployment manager for multi-cloud clusters
- Mixtures of private infrastructure and public cloud infrastructure
- Key management for clusters
- Applying updates to software
- Rolling back updates (versioning)
- Templating
- Tools
  - Helm (a package manager for Kubernetes)
  - Terraform

Labs
Kubernetes Lab #1

- Bookshelf Kubernetes Engine + Cloud Datastore + Cloud Storage (30 min)
Overall architecture

- 2 nodes, 3 replicas of a single Docker container image
- Network load-balancer to route requests to the external IP address on port 80 to one of the three replicas
Kubernetes Lab #1

- Ensure Kubernetes API is enabled
  - In web console, and select **Go to APIs Overview**
  - Search for Google Kubernetes Engine and ensure it is enabled
Create a Cluster

- Launch Cloud Shell
- List Compute Engine zones (which will run your containers)

```
$ gcloud compute zones list
```

- Set the region to the desired region:

```
$ gcloud config set compute/zone us-west1-b
```

- See configuration

```
$ gcloud config list
```
Not described in lab, but must be done

- Create a bucket to hold images uploaded by web site users

  - Via Google Cloud Shell
    
    ```
    gsutil mb -l <location> gs://$DEVSHHELL_PROJECT_ID
    ```

  - `gsutil` (Google Cloud Storage utility) command
  - `mb` = make bucket
  - Use `<location>` of *us-west1*
  - `gs://`
    - URI for all buckets (must be globally unique)
    - Use `<Project ID>` to uniquely label bucket
      - Note: you can use any name that is unique but the instructions assume you’ve named your bucket after your project ID
      - Get Project ID in Google Cloud Shell via `echo $DEVSHHELL_PROJECT_ID`

- Verify in console that bucket has been created

- Allow global read access to bucket
  
  ```
  gsutil defacl ch -u AllUsers:R gs://$DEVSHHELL_PROJECT_ID
  gsutil defacl set public-read gs://$DEVSHHELL_PROJECT_ID
  ```
• Create cluster called bookshelf with two nodes
• Set scope to cloud-platform so it can access services such as Cloud Storage and Cloud Datastore

```gcloud container clusters create bookshelf --scopes "https://www.googleapis.com/auth/userinfo.email","cloud-platform" --num-nodes 2```

```
wuchangfeng@lateral-array-175417:~$ gcloud container clusters create bookshelf \
> --scopes "https://www.googleapis.com/auth/userinfo.email","cloud-platform" \
> --num-nodes 2
Creating cluster bookshelf...done.
Created [https://container.googleapis.com/v1/projects/lateral-array-175417/zones/us-west1-b/clusters/bookshelf].
kubeconfig entry generated for bookshelf.
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>ZONE</th>
<th>MASTER_VERSION</th>
<th>MASTER_IP</th>
<th>MACHINE_TYPE</th>
<th>NODE_VERSION</th>
<th>NUM NODES</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookshelf</td>
<td>us-west1-b</td>
<td>1.6.7</td>
<td>35.197.43.71</td>
<td>n1-standard-1</td>
<td>1.6.7</td>
<td>2</td>
<td>RUNNING</td>
</tr>
</tbody>
</table>

wuchangfeng@lateral-array-175417:~$
- Examine code
  - Source Repositories=>Source code=>container-engine
  - Note: Alternative client libraries used to access Cloud Datastore for Container Engine version versus App Engine
• **Update** `config.py` and `bookshelf-frontend.yaml` with your project ID

```
sed -i s/your-project-id/$DEVSHELL_PROJECT_ID/ config.py
sed -i s/your-project-id/$DEVSHELL_PROJECT_ID/ bookshelf-frontend.yaml
```

• **Set container cluster to previously created one**

```
gcloud config set container/cluster bookshelf
```

• **Build Docker container from Dockerfile and code in current directory and register it in Google Cloud Registry**

```
docker build -t gcr.io/$DEVSHELL_PROJECT_ID/bookshelf .
```

• **Push it to the registry**

```
gcloud docker -- push gcr.io/$DEVSHELL_PROJECT_ID/bookshelf
```
- Kubernetes configuration (3 replicas, 1 service)

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: bookshelf-frontend
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: bookshelf
        tier: frontend
    spec:
      containers:
        - name: bookshelf-app
          # Docker image name (from previous step)
          image: gcr.io/cs410c-wuchang-201515/bookshelf
          # Always pull most recent version
          imagePullPolicy: Always
          # Specify process to start via PROCESSES variable for Docker
          env:
            - name: PROCESSES
              value: bookshelf
          ports:
            - containerPort: 8080
```

```
kind: Service
metadata:
  name: bookshelf-frontend
spec:
  type: LoadBalancer
  ports:
    - port: 80
  selector:
    app: bookshelf
    tier: frontend
```
• Get credentials for `kubectl`
  
  `gcloud container clusters get-credentials bookshelf`

• Use Kubernetes `kubectl` to deploy and scale on GKE
  
  `kubectl create -f bookshelf-frontend.yaml`

• Note that all containers run on Compute Engine instances

• But, configuration file portable
  
  • Can run on *any* other public cloud provider, locally on a laptop or on a private cloud, or even a hybrid cloud

• List pods
  
  `kubectl get pods`

• List services (with load-balancer)
  
  `kubectl get services bookshelf-frontend`

• Visit site via external IP address
• Add the book
  • Screenshot of site after book CPB200 Google BigQuery for Data Analysts is added with image (as described in Section 5)
• Show app in Container Engine
• Show nodes in Compute Engine
  • VM instances
  • Instance groups
  • Instance templates
• Show load balancer in Network Services
• Show External IP addresses in VPC network
• Show Docker image in Container Registry
• Clean up everything (to avoid charges)
Kubernetes Lab #1

- Bookshelf Kubernetes Engine + Cloud Datastore + Cloud Storage (30 min)
  - https://codelabs.developers.google.com/codelabs/cp100-container-engine/
AWS ECS Lab #1 (CS 510 only)

- Elastic Container Service 10-minute tutorial
Kubernetes Lab #2 alternative

• [https://codelabs.developers.google.com/codelabs/cloud-orchestrate-with-kubernetes](https://codelabs.developers.google.com/codelabs/cloud-orchestrate-with-kubernetes) (60 min)
  • From monolith to microservices
  • Separating auth, greeting, and front-end routing
Kubernetes Lab #3

- Managed instances vs. Kubernetes
  - Original monolithic app via Compute Engine cluster running nginx via managed instances
  - Microservices implementation using Kubernetes Engine (nginx, redis, mysql)
Create a Compute Engine Instance

- Create an instance with default settings:

```sh
$ gcloud compute instances create myinstance --zone us-central1-f
Created [...].
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>ZONE</th>
<th>MACHINE_TYPE</th>
<th>PREEMPTIBLE</th>
<th>INTERNAL_IP</th>
<th>EXTERNAL_IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>myinstance</td>
<td>us-central1-f</td>
<td>n1-standard-1</td>
<td></td>
<td>10.240.X.X</td>
<td>X.X.X.X</td>
</tr>
</tbody>
</table>

- Choose the zone that best suits your location/project.
- Note the `EXTERNAL_IP`.

- Enable port 80 in the firewall configuration:

```sh
$ gcloud compute firewall-rules create allow-80 --allow tcp:80
Created [...].
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>NETWORK</th>
<th>SRC_RANGES</th>
<th>RULES</th>
<th>SRC_TAGS</th>
<th>TARGET_TAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-80</td>
<td>default</td>
<td>0.0.0.0/0</td>
<td>tcp:80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Create a Compute Engine Instance

• To SSH into the instance from the command line

```
$ gcloud compute ssh myinstance
...
Do you want to continue (Y/n)? Y
...
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): [Hit Enter]
Enter same passphrase again: [Hit Enter]
...
yourusername@myinstance:~#
```

• Alternatively, you can use the WebUI to launch an SSH connection.
• Compute Engine > VM Instances, and clicking on SSH.
Install nginx

- Log into the newly created instance, and install nginx:

```
$ sudo su -
# apt-get update
# apt-get install -y nginx
# service nginx start
# exit
```
Install nginx

- Test that the server is running using `wget` from myinstance:

```
$ wget -q -O - localhost:80
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
</body>
</html>
```
Install nginx

- Find the external IP for your instance by listing your instances either via the web UI:

  - Or by exiting from the SSH connection, and using the command line:

    ```
    $ gcloud compute instances list
    
    NAME          ZONE             MACHINE_TYPE   PREEMPTIBLE INTERNAL_IP
    EXTERNAL_IP   STATUS
    myinstance    us-central1-f    n1-standard-1  10.240.0.2
    104.155.42.166 RUNNING
    ```
Install nginx

- Navigate to http://EXTERNAL_IP where EXTERNAL_IP is the public IP from the previous slide. You should see the nginx page.

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.
Create a Cluster of Servers

- Create an instance template using the startup script:

```bash
$ gcloud compute instance-templates create nginx-template \
    --metadata-from-file startup-script=startup.sh
Created [...].
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>MACHINE_TYPE</th>
<th>PREEMPTIBLE</th>
<th>CREATION_TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-template</td>
<td>n1-standard-1</td>
<td></td>
<td>2015-11-09T08:44:59.007-08:00</td>
</tr>
</tbody>
</table>

- Create a target pool:

```bash
$ gcloud compute target-pools create nginx-pool
Created [...].
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>REGION</th>
<th>SESSION_AFFINITY</th>
<th>BACKUP</th>
<th>HEALTH_CHECKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-pool</td>
<td>us-central1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Create a Cluster of Servers

- Finally, create an instance group using the template:

```sh
$ gcloud compute instance-groups managed create nginx-group \
  --base-instance-name nginx \
  --size 2 \
  --template nginx-template \
  --target-pool nginx-pool
Created [...].
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>ZONE</th>
<th>BASE_INSTANCE_NAME</th>
<th>SIZE</th>
<th>TARGET_SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-group</td>
<td>us-central1-f</td>
<td>nginx</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>nginx-group</td>
<td>nginx-template</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Create a Network Load Balancer

- Create a network load balancer targeting our instance group

```bash
$ gcloud compute forwarding-rules create nginx-lb \
   --ports 80 \
   --target-pool nginx-pool
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>REGION</th>
<th>IP_ADDRESS</th>
<th>IP_PROTOCOL</th>
<th>TARGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-lb</td>
<td>us-central1</td>
<td>104.155.48.184</td>
<td>TCP</td>
<td>us-central1/targetPools/nginx-pool</td>
</tr>
</tbody>
</table>

- Visit the load balancer from the browser 
  `http://IP_ADDRESS/` where `IP_ADDRESS` is the address shown as the result of running the previous command.
Clean up cluster

- Don't forget to shut down your cluster, otherwise they'll keep running and accruing costs!
- The following commands will delete the Google Compute Engine instances, Instance Group, Targeting Group, and the Load Balancer:

```bash
$ gcloud compute forwarding-rules delete nginx-lb
$ gcloud compute instance-groups managed delete nginx-group
$ gcloud compute target-pools delete nginx-pool
$ gcloud compute instance-templates delete nginx-template
$ gcloud compute instances delete myinstance
$ gcloud compute instances delete nginx
$ gcloud compute firewall-rules delete allow-80
```
Create a Kubernetes Cluster

- Don't forget to set the default zone and region:
  
  ```
  $ gcloud config set compute/zone us-central1-f
  $ gcloud config set compute/region us-central1
  ```

- Use Container Engine to create a cluster:
  
  ```
  $ gcloud container clusters create guestbook --
  num-nodes 3
  ```

- You can see the newly created instances in the
  
  - Google Compute Engine > VM Instances page.
Create a Kubernetes Cluster

- Deploy nginx in Kubernetes:

```bash
$ kubectl run nginx --image=nginx --replicas=3
```

- You can see that each nginx pod is now running in a different node (virtual machine):

```bash
$ kubectl get pods -o wide
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
<th>NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-ffssc</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
<td>gke-demo-2-43558313-node-6476</td>
</tr>
<tr>
<td>demo-2-43558313-node-sgve</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nginx-nk1ok</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
<td>gke-demo-2-43558313-node-mg69</td>
</tr>
<tr>
<td>demo-2-43558313-node-hswk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nginx-x86ck</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
<td>gke-demo-2-43558313-node-xcrn</td>
</tr>
<tr>
<td>demo-2-43558313-node-wskh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Create a Kubernetes Cluster

- Expose the NGINX cluster as an external service:
  
  ```bash
  $ kubectl expose deployment nginx --port=80 --target-port=80 --type=LoadBalancer
  ```

- Find the network load balancer address:
  
  ```bash
  $ kubectl get service nginx
  NAME  CLUSTER_IP   EXTERNAL_IP   PORT(S)    SELECTOR     AGE
  nginx 10.X.X.X     X.X.X.X       80/TCP     run=nginx    1m
  ```

- Navigate to `http://EXTERNAL_IP` where `EXTERNAL_IP` is the public IP from the previous slide. You should see the Nginx page.

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.
Delete the Kubernetes Cluster

- Don’t forget to delete the service after you’re done:
  
  ```
  $ kubectl delete service nginx
  service "nginx" deleted
  ```

- Next, delete the replication controller. This will subsequently delete the pods (all of the NGINX instances) as well:
  
  ```
  $ kubectl delete deployment nginx
  deployment "nginx" deleted
  ```
Get the Guestbook source

- Start by cloning the github repository for the Guestbook application:

  $ git clone https://github.com/saturnism/spring-boot-docker

- Move into the kubernetes examples directory:

  $ cd spring-boot-docker/examples/kubernetes

- First create a pod using kubectl, the Kubernetes CLI:

  $ kubectl create -f redis-pod.yaml
Turn Up the Redis Pod and Service

- Check that a Redis instance is running:

  ```
  $ kubectl get pods
  NAME       READY     STATUS    RESTARTS   AGE
  redis      1/1       Running   0          20s
  ```

- Create the Redis service:

  ```
  $ kubectl create -f redis-service.yaml
  ```

- And check it:

  ```
  $ kubectl get services
  NAME          LABELS                                         SELECTOR
  IP(S)          PORT(S)
  kubernetes     component=apiserver,provider=kubernetes       <none>
  10.107.240.1   443/TCP
  redis          name=redis,role=master,visualize=true
                  name=redis,role=master
  10.107.254.132 6379/TCP
Turn Up the MySQL Pod and Service

- Start by cloning the github repository for the Guestbook application

```
$ gcloud compute disks create mysql-disk --size 200GB
Created [...].
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>ZONE</th>
<th>SIZE_GB</th>
<th>TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql-disk</td>
<td>us-central1-f</td>
<td>200</td>
<td>pd-standard</td>
<td>READY</td>
</tr>
</tbody>
</table>

- Deploy both the MySQL Pod and the Service with a single command:

```
$ kubectl create -f mysql.yaml -f mysql-service.yaml
```
Deploying the Microservices

- Deploy the Hello World Replication Controller:
  
  ```bash
  $ kubectl create -f helloworldservice-controller-v1.yaml
  ```

- See the replicas with:
  
  ```bash
  $ kubectl get rc
  ```

- Also see the pods running with:
  
  ```bash
  $ kubectl get pods
  ```

- Look at each pod's log output by running:
  
  ```bash
  $ kubectl logs -f helloworldservice-controller-v1-XXXXXX
  ```
Resizing a Replication Controller

- Scale the number of replicas of our Hello World controller by running:
  ```bash
  $ kubectl scale rc helloworldui-controller-v1 --replicas=12
  ```

- See that the replication controller has been updated:
  ```bash
  $ kubectl get rc
  ```
Kubernetes Lab #3

- Managed instances vs. Kubernetes
  - https://codelabs.developers.google.com/codelabs/cloud-compute-kubernetes (130 min)