Lecture 20: Operations - Scaling

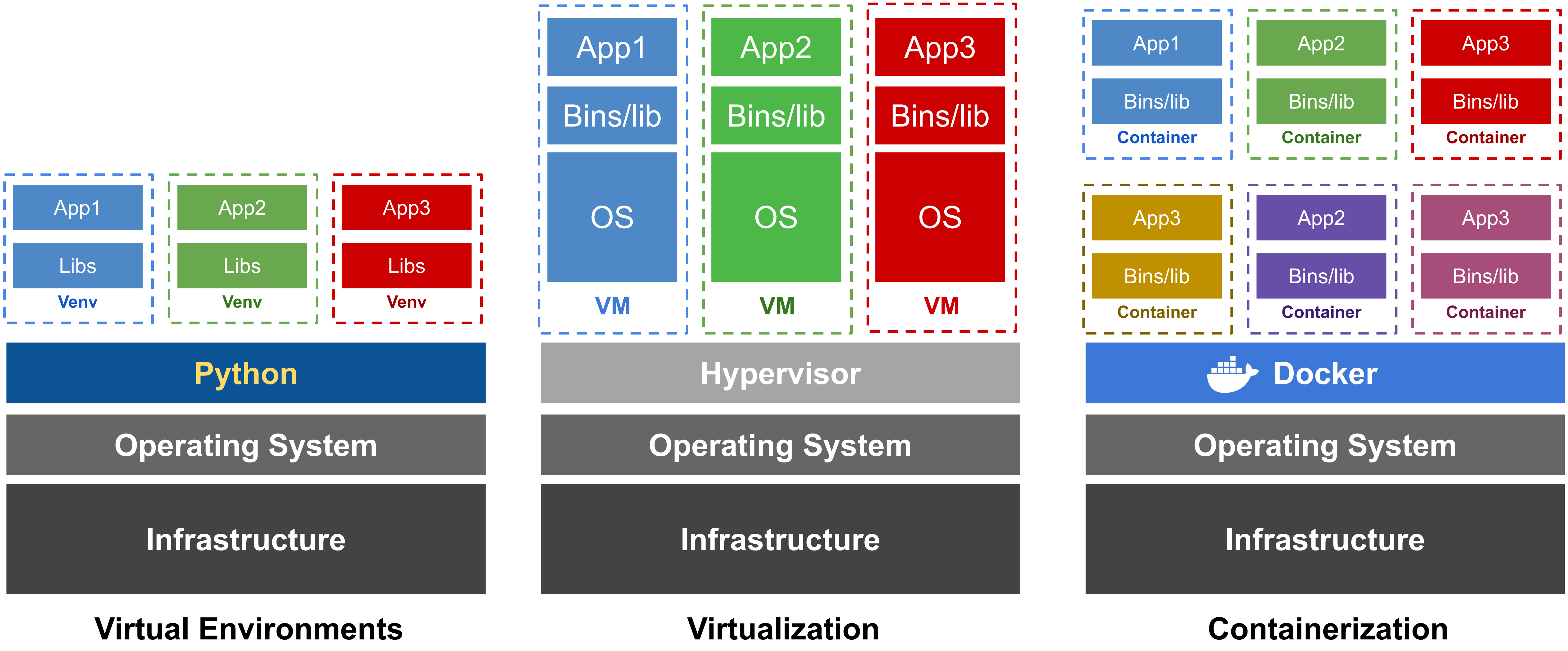
AC215

Pavlos Protopapas
SEAS/ Harvard

Outline

1. Recap
2. Motivation
3. Introduction to Kubernetes
4. Tutorial: Deploying a Kubernetes Cluster
5. Advantages of using Kubernetes

Recap



App1
Libs
Venv

App2
Libs
Venv

App3
Libs
Venv

Python

Operating System

Infrastructure

Virtual Environments

App1
Bins/lib
OS
VM

App2
Bins/lib
OS
VM

App3
Bins/lib
OS
VM

Hypervisor

Operating System

Infrastructure

Virtualization

App1
Bins/lib
Container

App2
Bins/lib
Container

App3
Bins/lib
Container

App3
Bins/lib
Container

App2
Bins/lib
Container

App3
Bins/lib
Container

Docker

Operating System

Infrastructure

Containerization

Recap

Virtual Environment

Pros: remove complexity
Cons: does not isolate from OS

Virtual Machines

Pros: isolate OS guest from host
Cons: intensive use hardware

Containers

Pros: lightweight
Cons: issues with security, scalability, and control

Recap

Virtual Environment

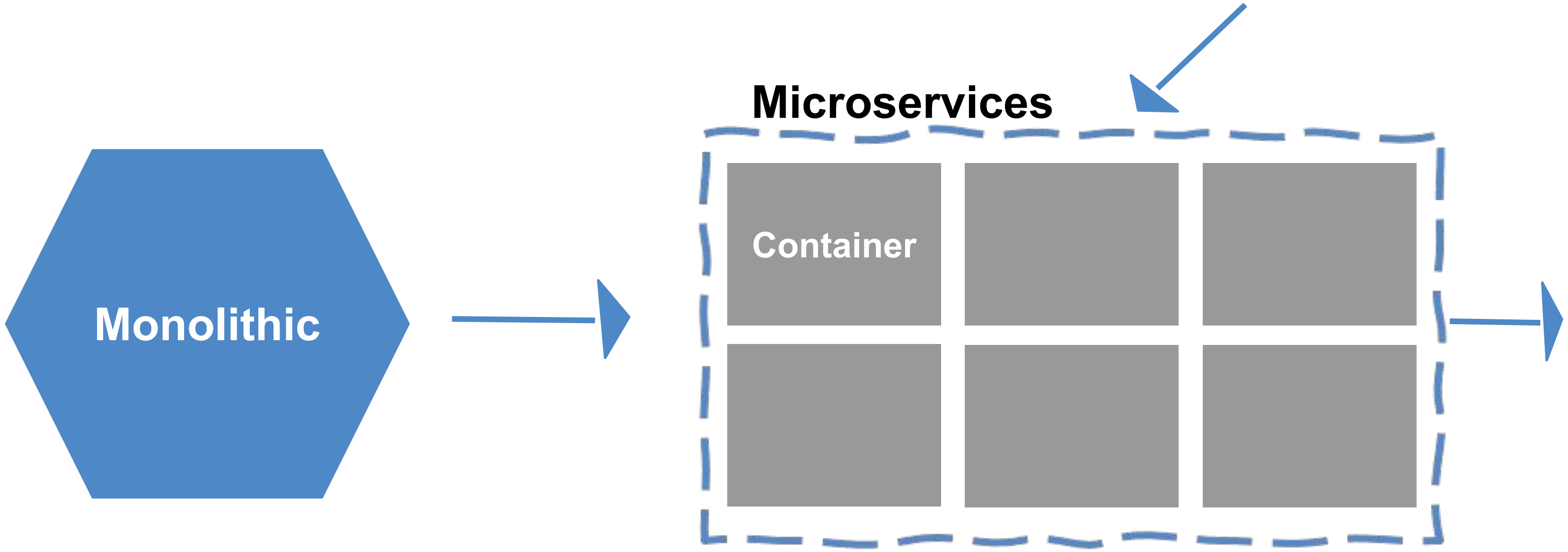
Pros: remove complexity
Cons: does not isolate from OS

Virtual Machines

Pros: isolate OS guest from host
Cons: intensive use hardware

Containers

Pros: lightweight
Cons: issues with security, scalability, and control



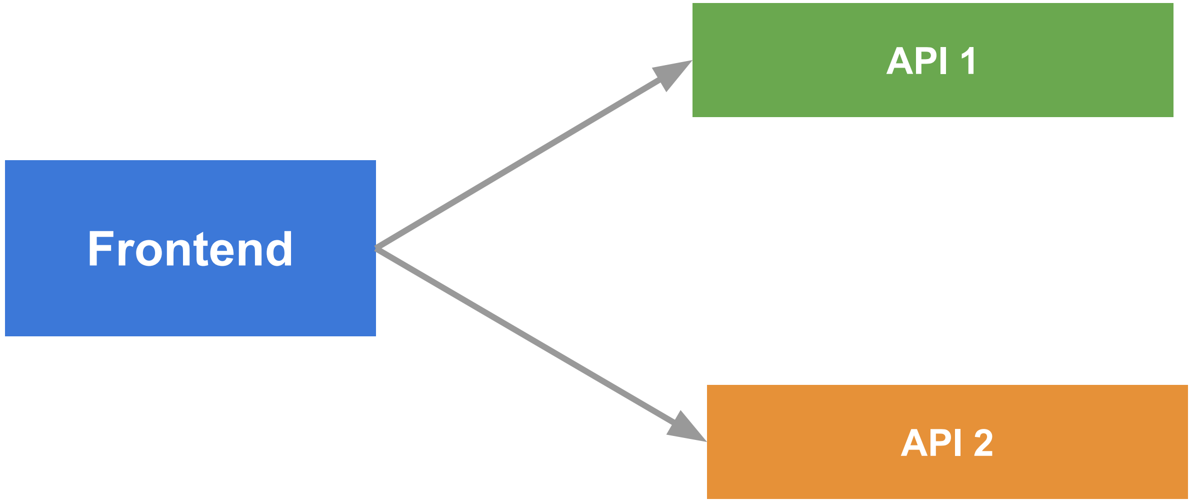
How to manage microservices?

Outline

1. Recap
2. **Motivation**
3. Introduction to Kubernetes
4. Tutorial: Deploying a Kubernetes Cluster
5. Advantages of using Kubernetes

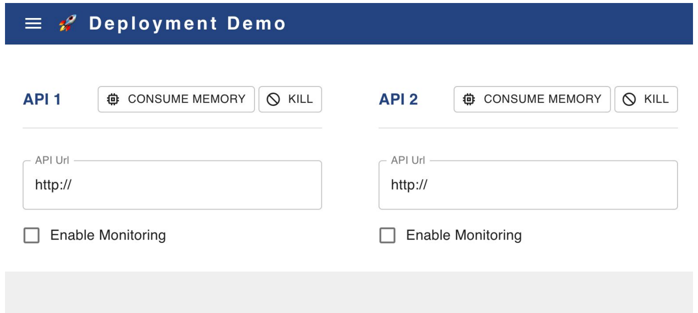
Motivation

Pavlos wants an app with 1 frontend & 2 backends



Motivation - 3 Containers in 1 VM

Support builds and deploys the app with the following architecture



Deployment Demo

API 1

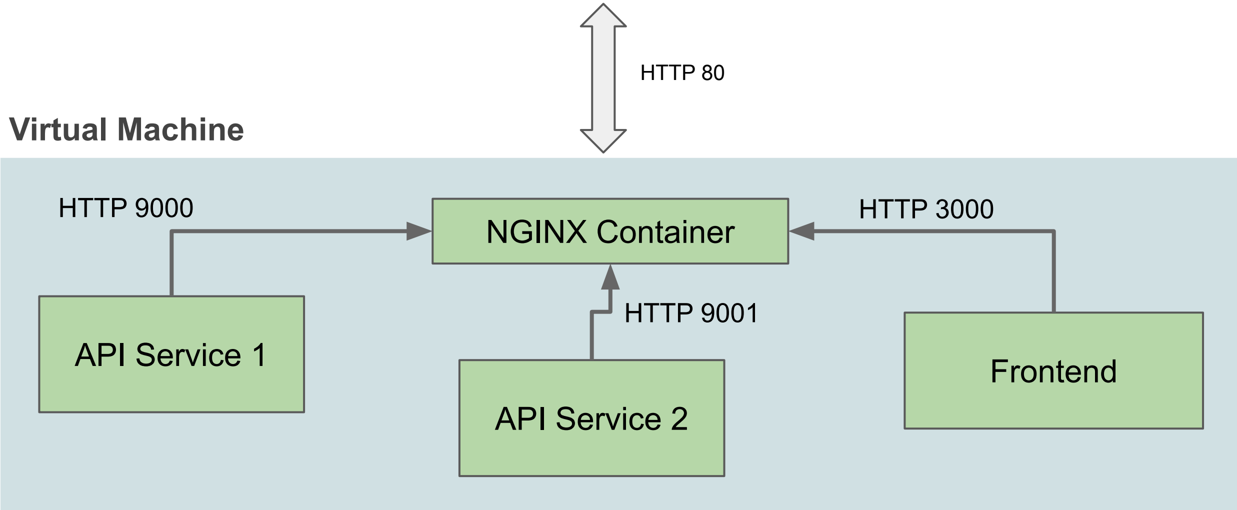
API Url: http://

□ Enable Monitoring

API 2

API Url: http://

□ Enable Monitoring



Motivation - 3 Containers in 1 VM

Demo… [3 Containers in 1 VM]

Motivation - 3 Containers in 1 VM

Container Crashes

Pavlos must contact support for resolution.

Support Actions:

Access the server via SSH.

- Perform the following fixes:
- Restart the container to reset memory.
- Relaunch a terminated container.

Motivation - 3 Containers in 3 VM

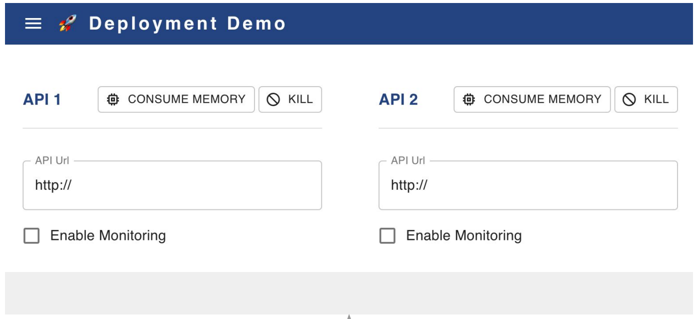
Pavlos' Request to *Support*

Can we deploy the app across multiple servers?

This way, if one server goes down, I'll have a backup to rely on."

Motivation - 3 Containers in 3 VM

Support deploys the app on to 3 servers with backup apis



API 1 CONSUME MEMORY | KILL

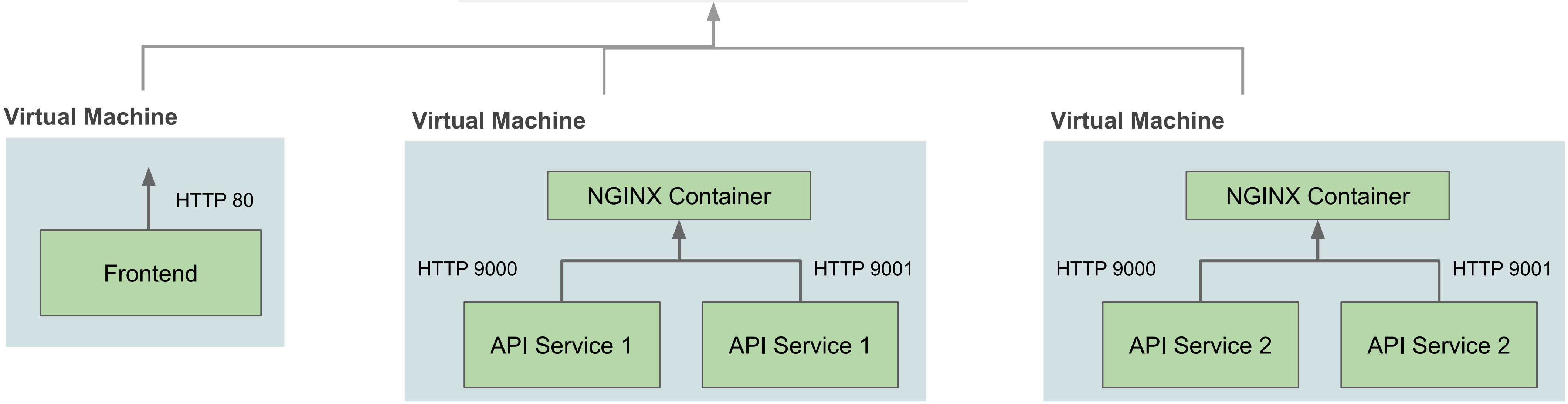
API Url: http://

☐ Enable Monitoring

API 2 CONSUME MEMORY | KILL

API Url: http://

☐ Enable Monitoring



Motivation - 3 Containers in 3 VM

Demo… [3 Containers in 3 VMs]

Motivation - 3 Containers in 3 VM

Problems:

- When container crashes, Pavlos can switch to backup API manually
- *Support* SSHs into server and fix when available:
    - Memory reset with container restart
    - Startup a killed container

Motivation - Kubernetes

Pavlos' Question to Support

- Can we automate:
  - Failovers
  - Load balancing
  - Scaling
  - And other key processes?"

Kubernetes to the rescue...

Kubernetes (K8s) to the Rescue

- K8s is an orchestration tool for **managing distributed containers** across a cluster of nodes (VMs).
- The word Kubernetes comes from the ancient Greek word **kubernḗtēs**, which means helmsman or pilot. The name is a reference to the role of a **helmsman**, who steers a ship and maintains a steady course.
- Kubernetes was announced by Google on June 6, 2014. The project was conceived and created by Google employees Joe Beda, Brendan Burns, and Craig McLuckie.

Kubernetes (K8s) to the Rescue

- Kubernetes (K8s) is made up of building blocks that help deploy and scale applications based on CPU, memory, or custom metrics.

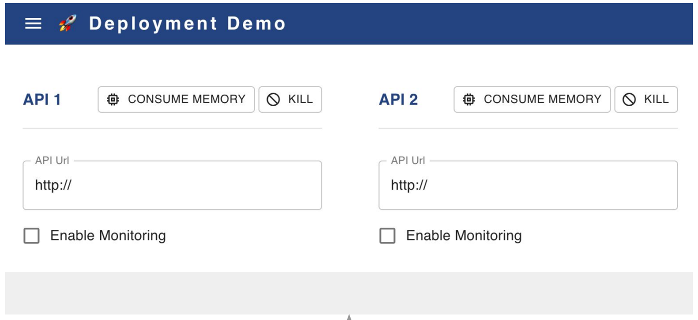
- K8s itself follows a **primary-replica architecture** with components that govern an individual node and others part of the **control plane**

- Core concepts in Kubernetes include pods, services and deployments.

- K8s **users define rules** for how container management should occur, and then K8s handles the rest!

Kubernetes to the Rescue

Support deploys the app on to 3 k8s clusters with 2 nodes each



Deployment Demo

API 1 CONSUME MEMORY KILL

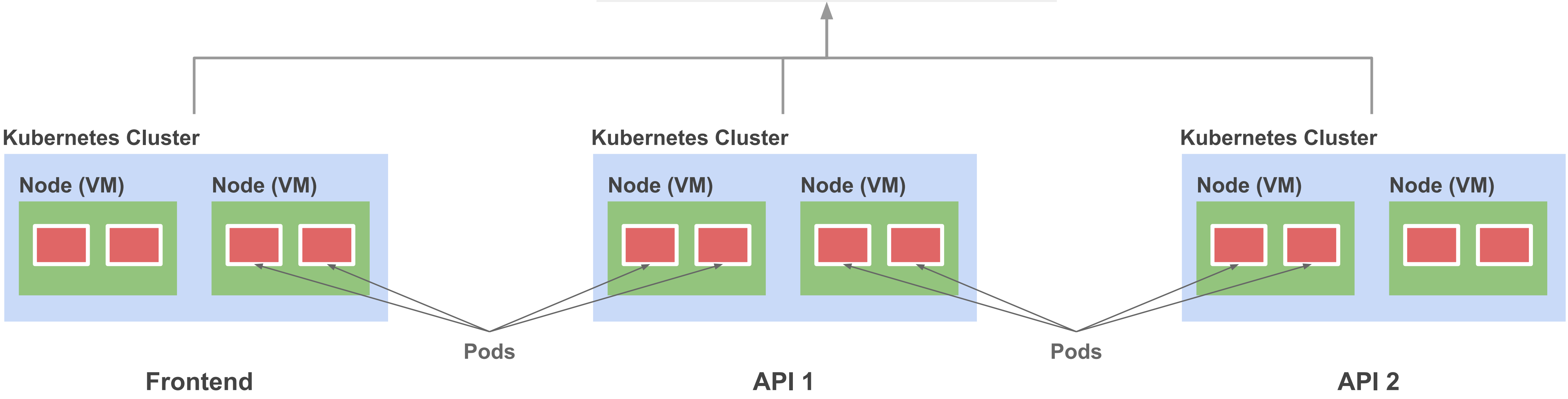
API Url
http://

□ Enable Monitoring

API 2 CONSUME MEMORY KILL

API Url
http://

□ Enable Monitoring



Kubernetes Cluster
Node (VM)
Node (VM)
Frontend

Kubernetes Cluster
Node (VM)
Node (VM)
API 1

Kubernetes Cluster
Node (VM)
Node (VM)
API 2

Pods

Pods

Kubernetes to the Rescue

Demo… [Kubernetes Cluster]

Kubernetes

Pavlos requests on automation:

✓ • Failovers

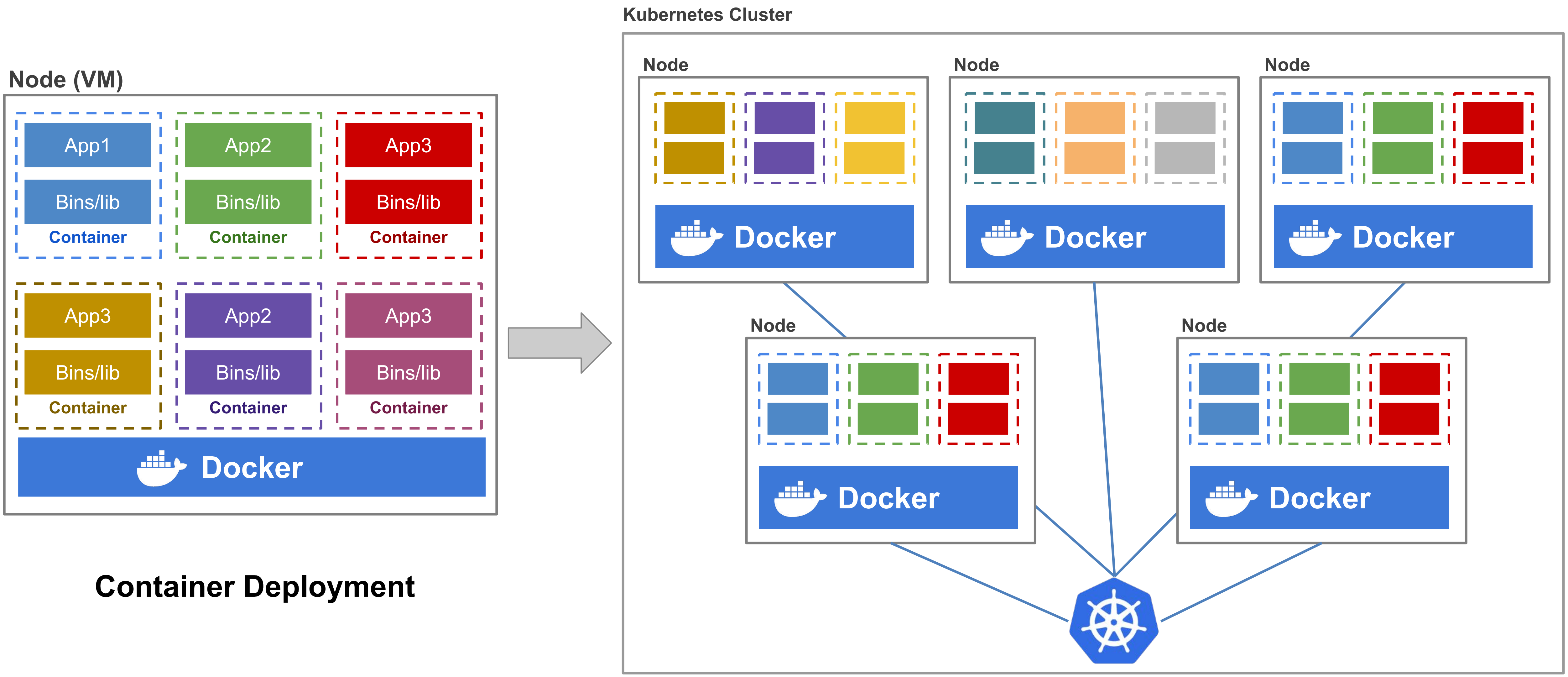
✓ • Load balancing

✓ • Scaling

Outline

1. Recap
2. Motivation
3. **Introduction to Kubernetes**
4. Tutorial: Deploying a Kubernetes Cluster
5. Advantages of using Kubernetes

Container vs Kubernetes Deployment



Container Deployment

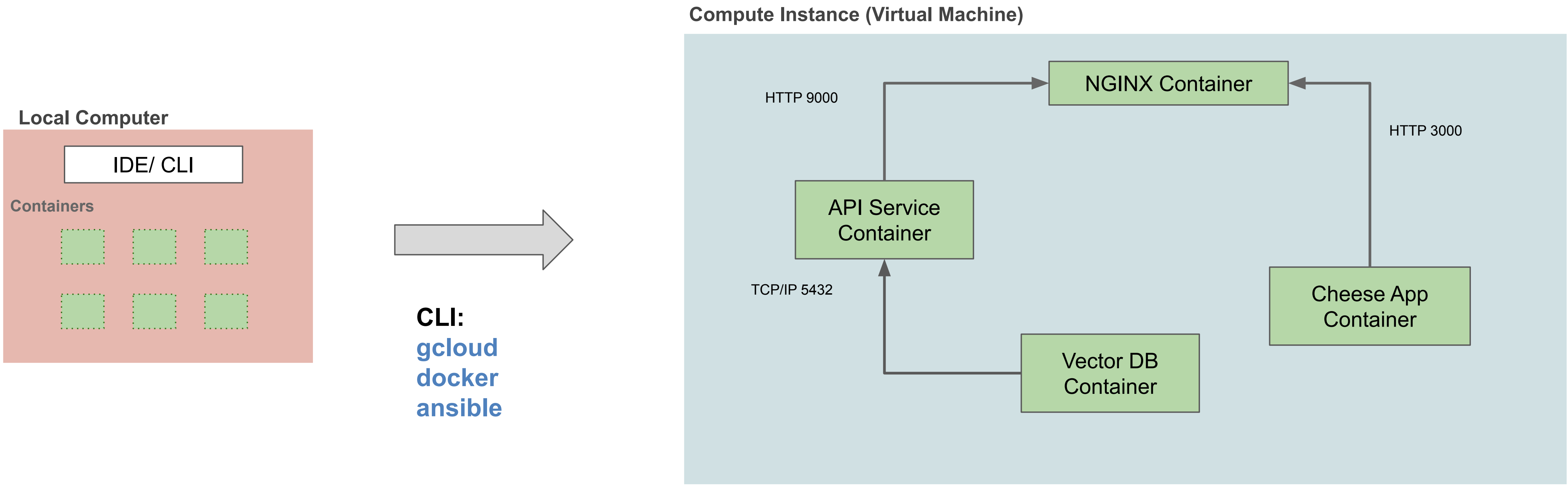
Kubernetes Deployment

Why Kubernetes?

- **Automating** and **Management** of Microservices
- **Bridging** Application Deployment & Deployment (Dev + Ops)
- **Standardizing** Cloud Deployments
- Daily **Management** of Applications

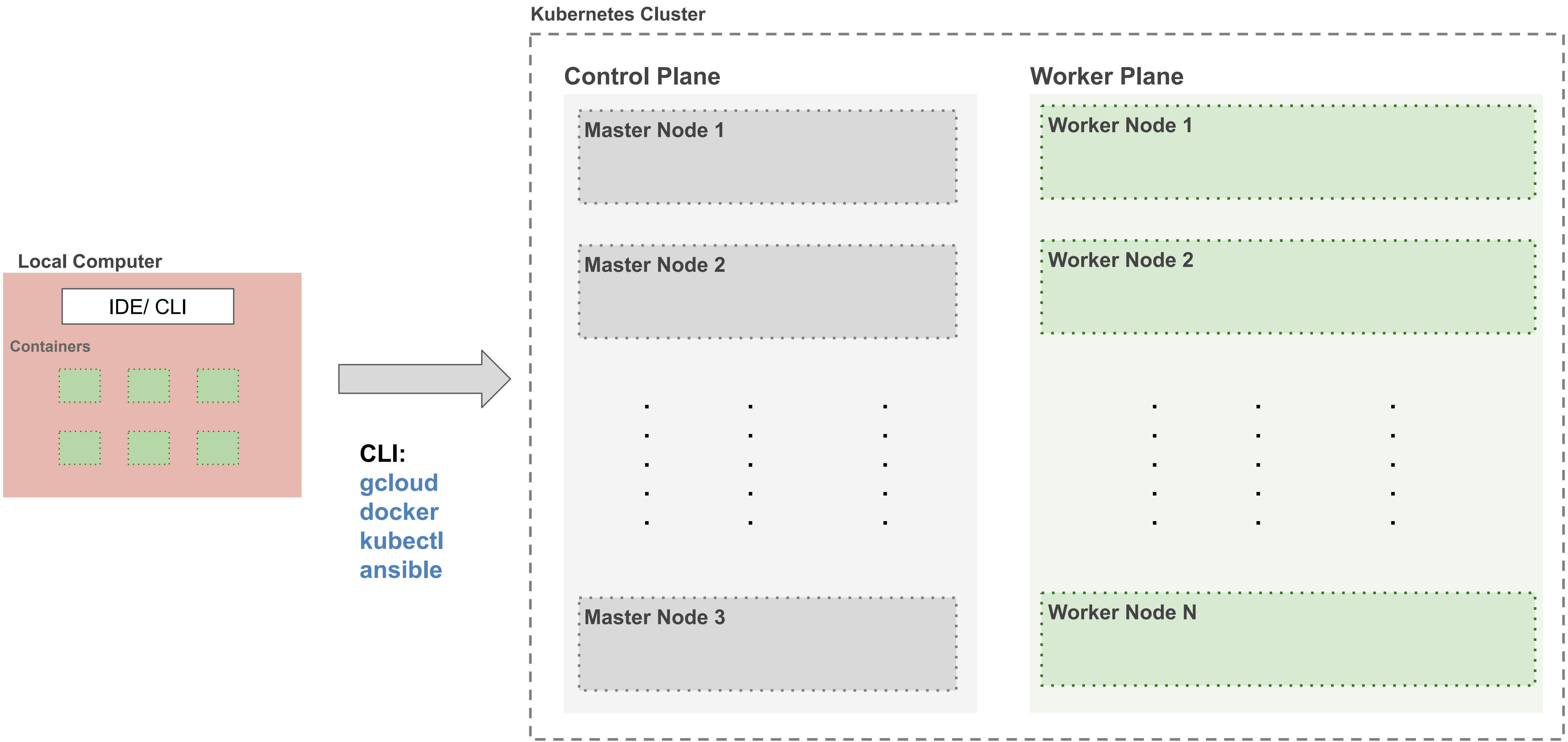
How do we build with Kubernetes?

Remember the Cheese App Architecture:



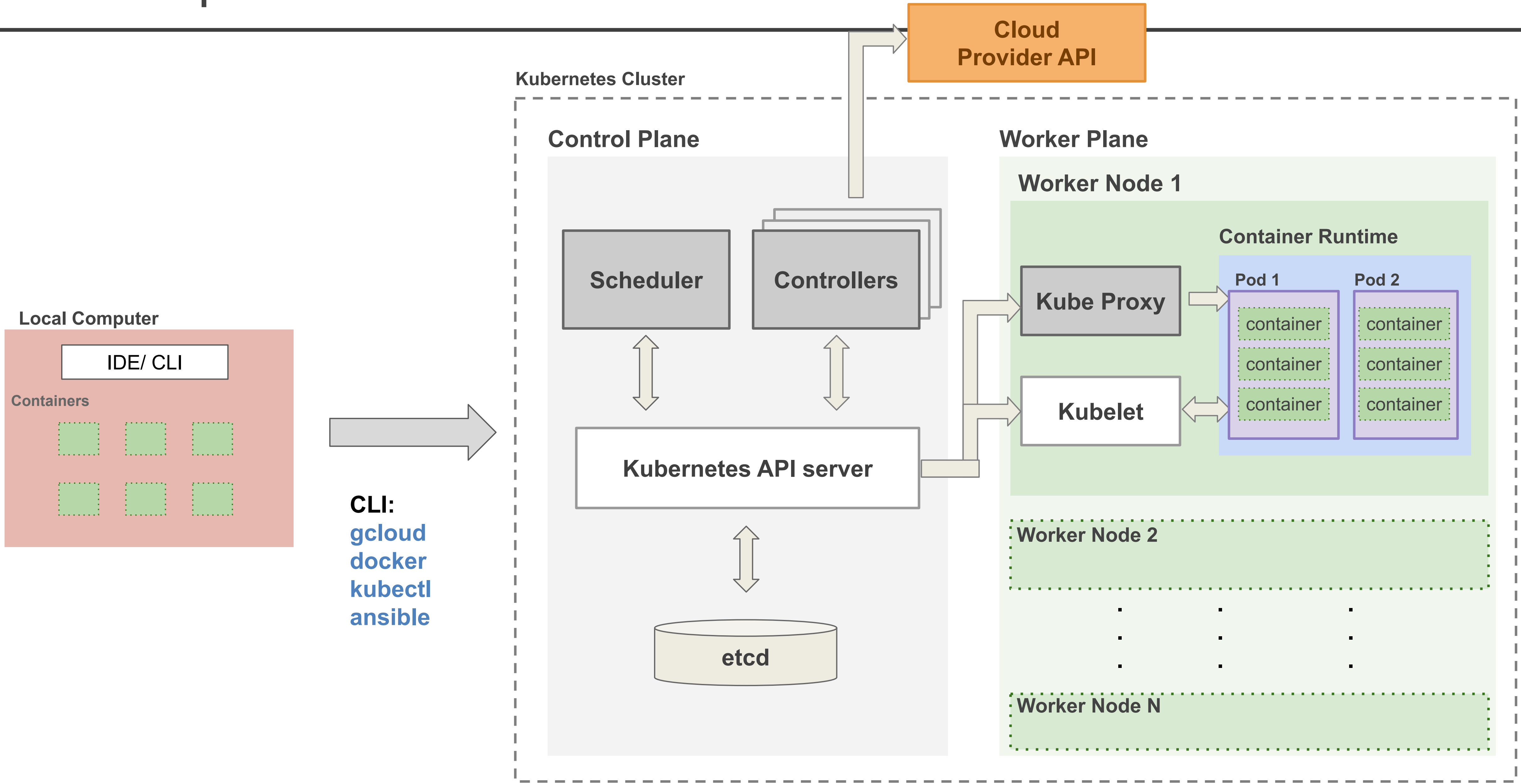
CLI:
gcloud
docker
ansible

K8s Components & Architecture



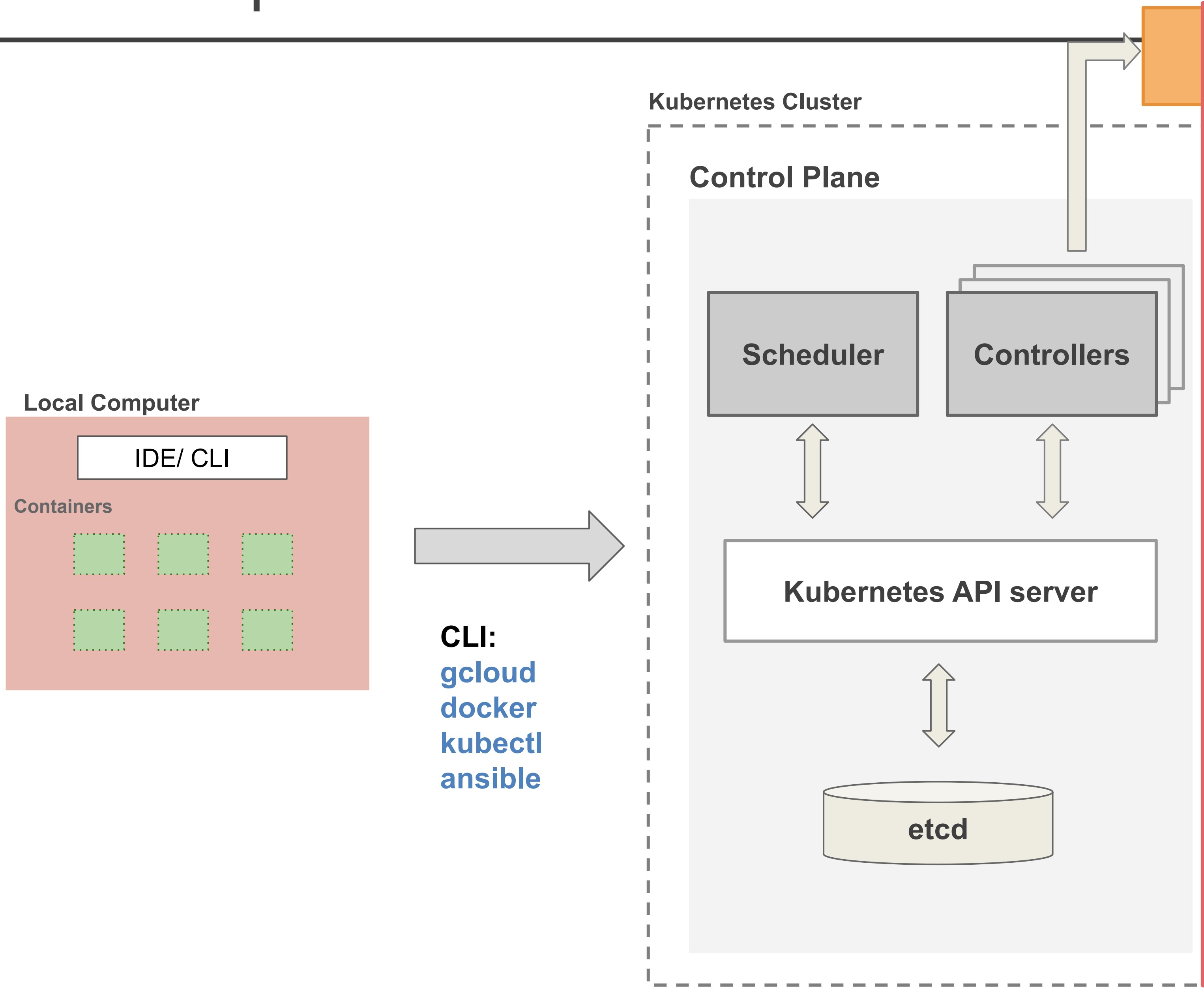
CLI:
gcloud
docker
kubectl
ansible

K8s Components & Architecture



CLI:
gcloud
docker
kubectl
ansible

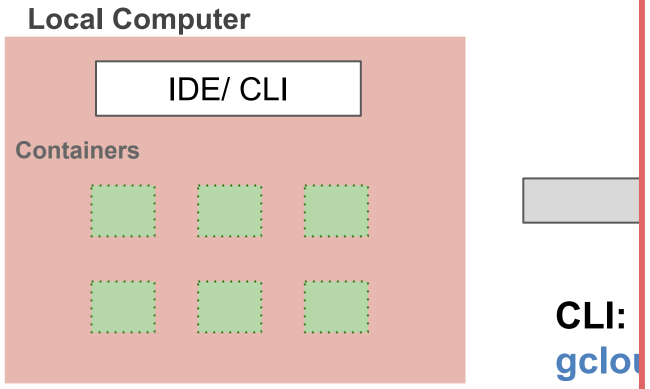
K8s Components & Architecture



The control plane has:

- **API server** contains various methods to directly access the Kubernetes
- **etcd** works as backend for service discovery that stores the cluster's state and its configuration
- **Scheduler** assigns applications to each worker node
- **Controller manager**:
  - Keeps track of worker nodes
  - Handles node failures and replicates if needed
  - Provide endpoints to access the application from the outside world
  - Communicates with cloud provide regarding resources such as nodes and IP addresses

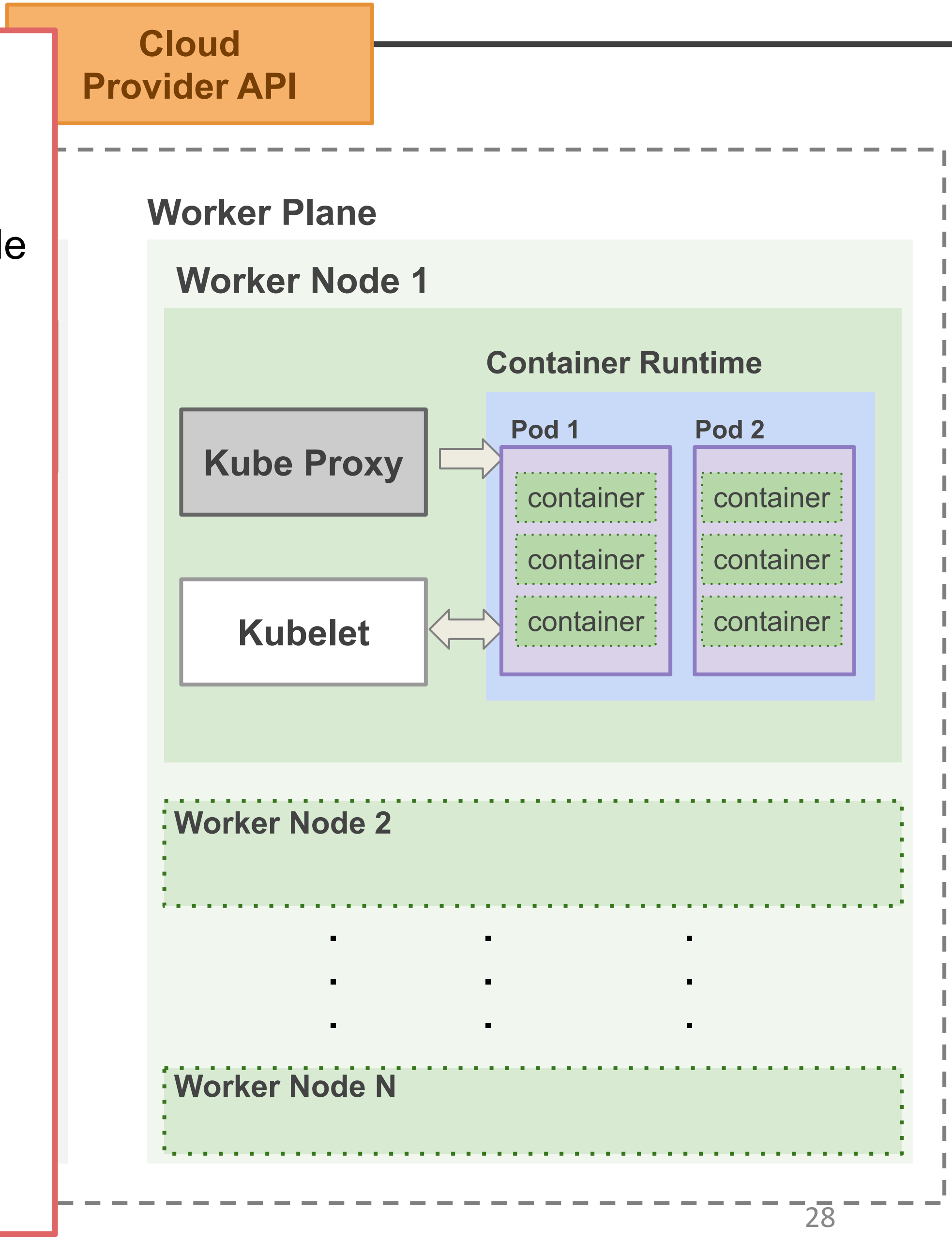
K8s Components & Architecture



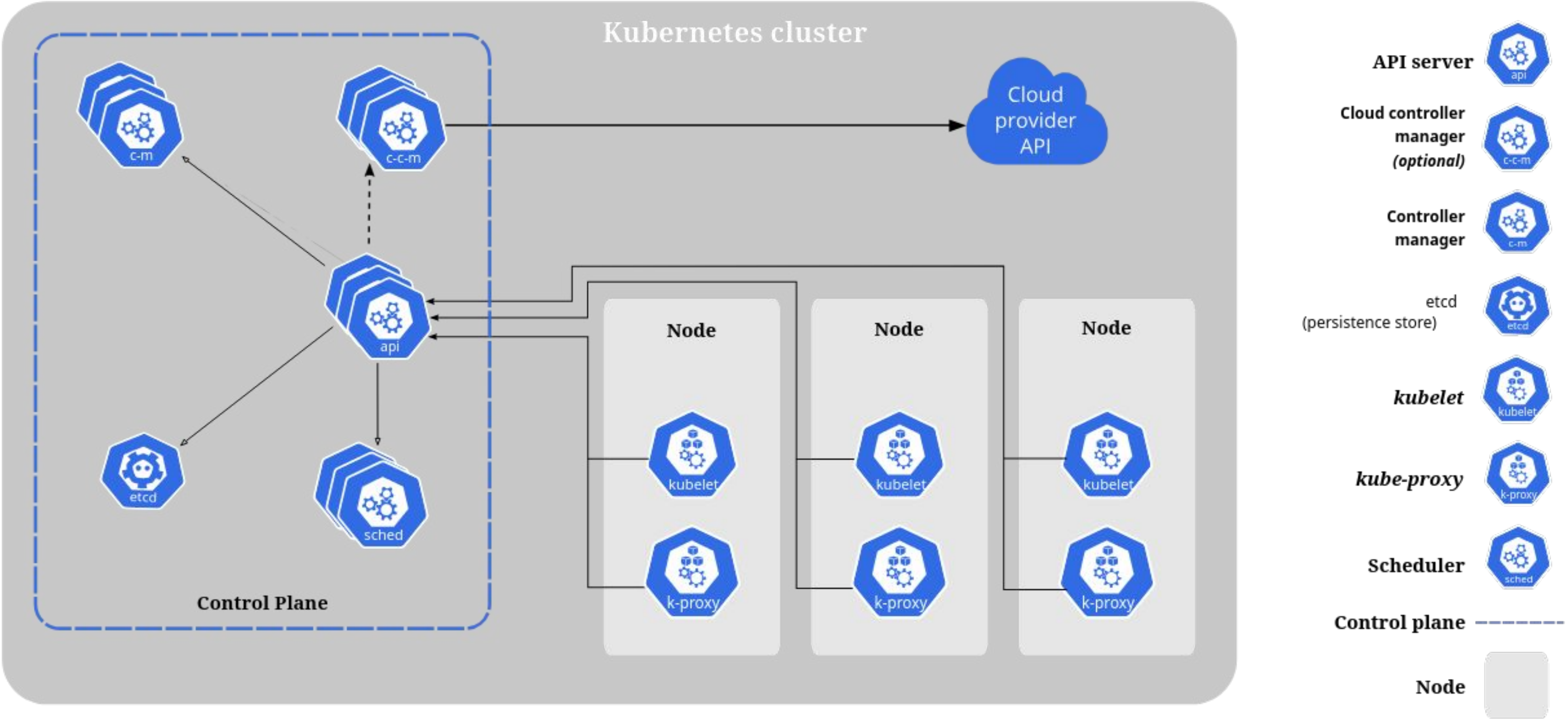
CLI:
gclou
dock
kube
ansil

The worker node consists of:

- **Kubelet** talks to the API server and manages containers on its node
- **Kube Proxy** load-balances network traffic between application components and the outside world
- **Container Runtime**: In our case this will be Docker. The runtime host Pods which run container instances



K8s Architecture

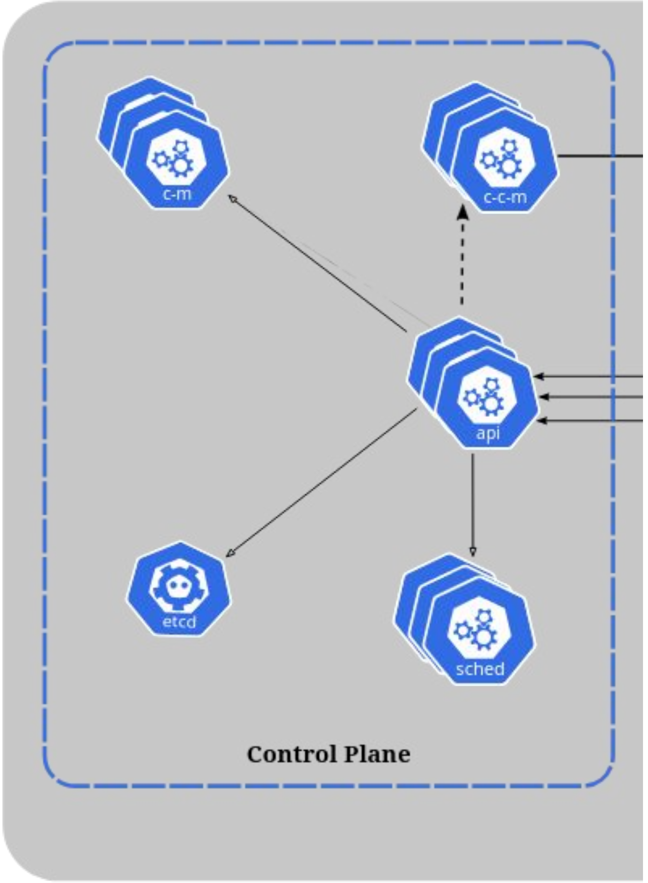


K8s Architecture – Control Pane

API Server – The core component server that exposes the Kubernetes HTTP API

Scheduler – Looks for Pods not yet bound to a node, and assigns each Pod to a suitable node.

Etcd – Consistent and highly-available key value store for all API server data

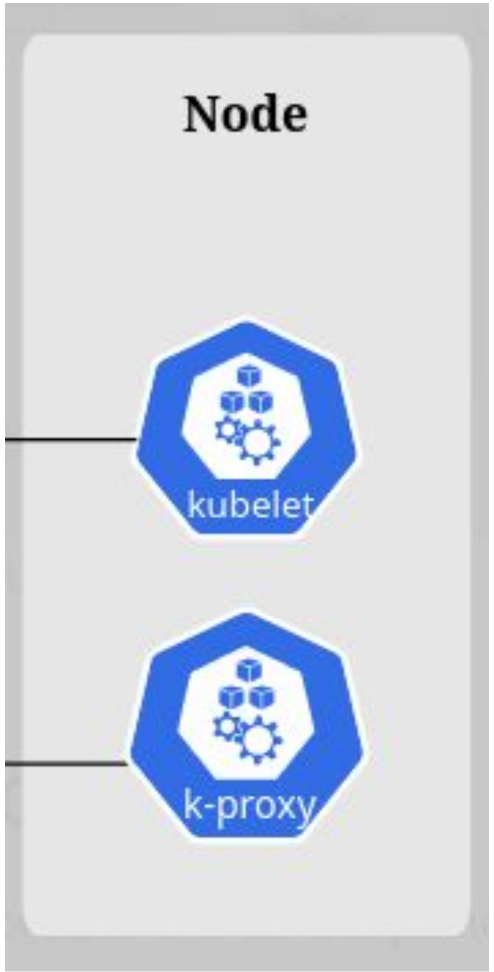


K8s Architecture – Node

Kubelet – Ensures that Pods are running, including their containers.

kube-proxy – Maintains network rules on nodes to implement Services.

Container runtime – Software responsible for running containers. e.g. Docker

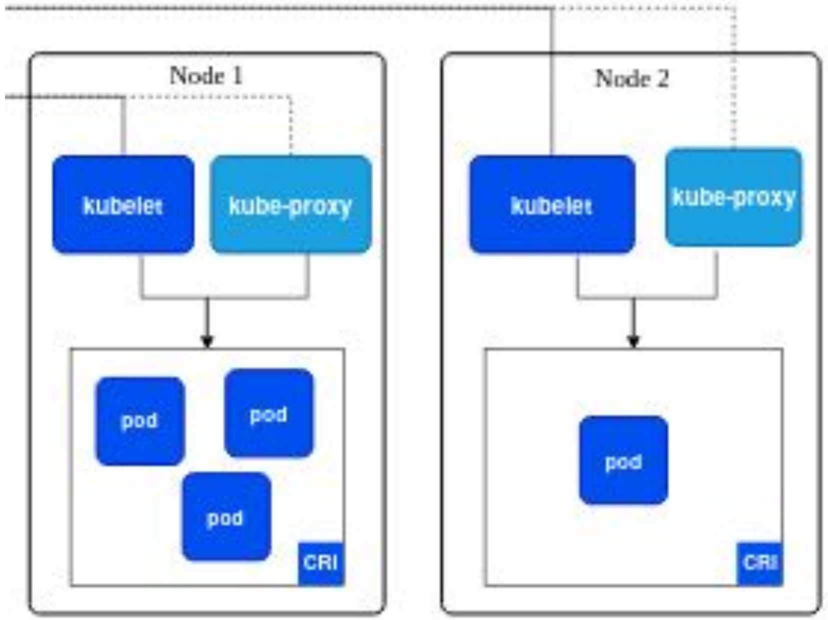


K8s Architecture – Pod

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.

A **Pod** is a group of containers that share resources like storage and networking. It's like a virtual host for one or more closely related containers that work together.

Pods are *ephemeral*. Whenever they die if needed they're replaced by a new pod.

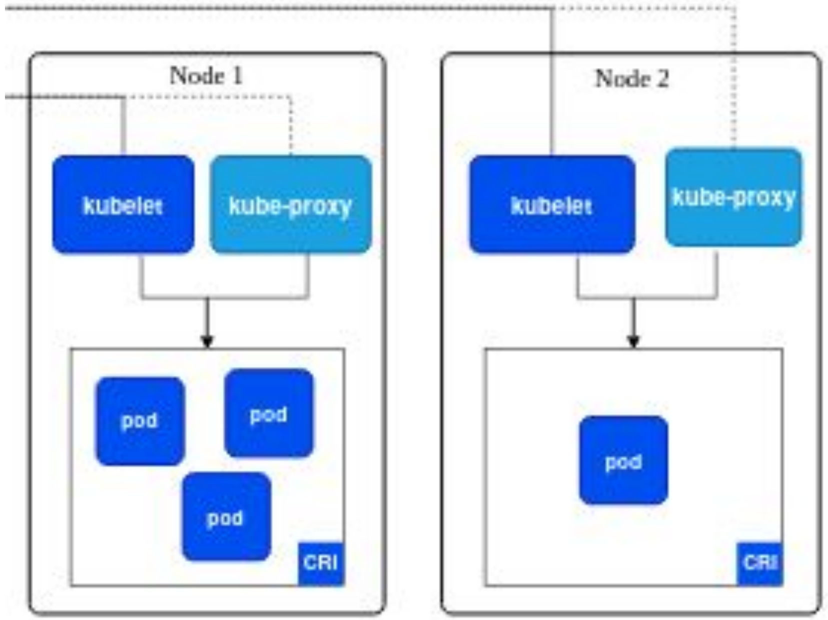


K8s Architecture – Pod

K8s Pods come in two main use cases.

- **Single-container Pods:** Most common use case. A Pod wraps around a single container, simplifying management.

- **Multi-container Pods:** Advanced use case. Multiple tightly coupled containers share resources and form a single unit, ideal for specific applications.

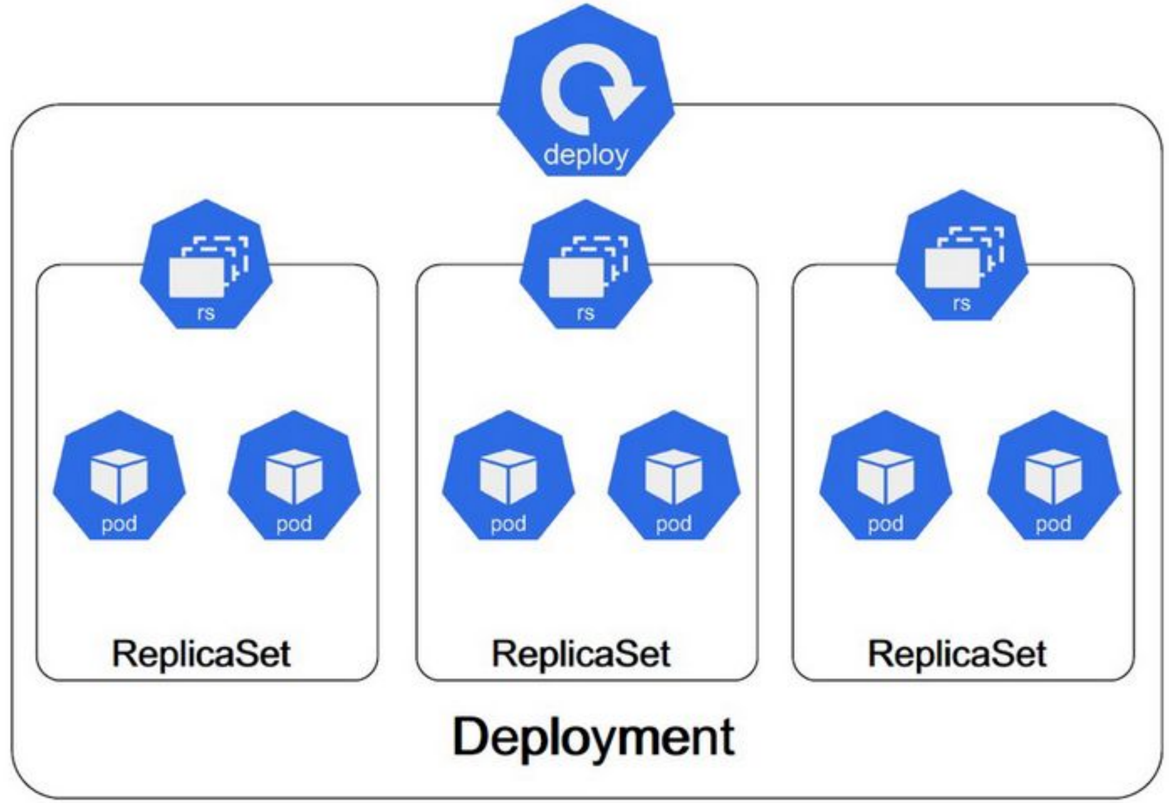


K8s Architecture – Deployments

A **Deployment** manages multiple Pods to run a stateless application.

Key features:

- Declarative updates for **Pods** and **ReplicaSets**
- Automatically adjusts actual state to match desired state
- Supports creating new ReplicaSets or adopting existing resources



Hierarchical structure of Deployment, ReplicaSet, and Pod (adapted from official documentation of Kubernetes (https://kubernetes.io/docs/concepts/workloads/controllers/, accessed on 10 January 2023)).

K8s Architecture – Deployments

The following are typical use cases for **Deployments**:

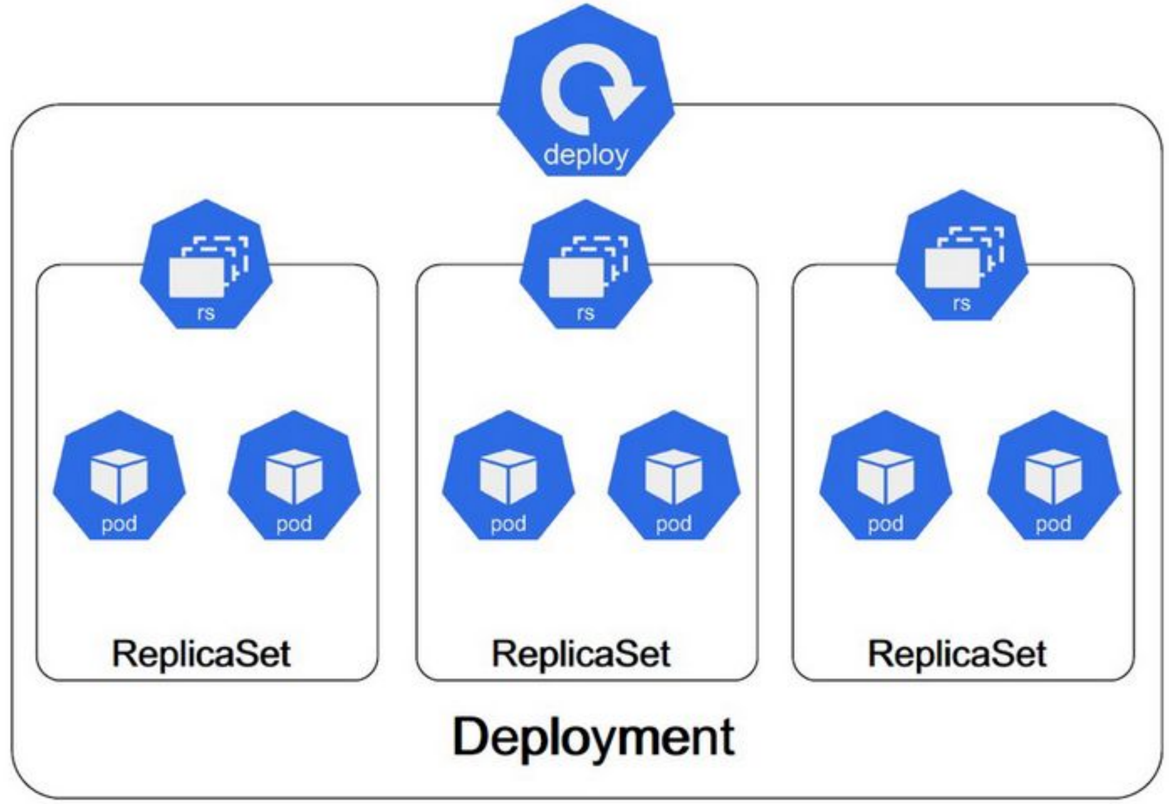
- Create a **Deployment**, which automatically rolls out a **ReplicaSet** and creates Pods.
- Create a new **ReplicaSet** and manage the transition
- Check rollout status for success.
- Roll back to a previous revision if unstable.
- Scale up the **Deployment** to handle increased load

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
        app: nginx
spec:
  replicas: 3
  selector:
        matchLabels:
        app: nginx
  template:
        metadata:
        labels:
        app: nginx
        spec:
        containers:
        - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

K8s Architecture – ReplicaSet

A **ReplicaSet's** purpose is to maintain a stable set of replica Pods running at any given time.

Usually, you define a Deployment and let that Deployment manage **ReplicaSets** automatically.



Hierarchical structure of Deployment, ReplicaSet, and Pod (adapted from official documentation of Kubernetes (https://kubernetes.io/docs/concepts/workloads/controllers/, accessed on 10 January 2023)).

K8s Architecture – ReplicaSet

A **ReplicaSet's** is often used to guarantee the availability of a specified number of identical Pods.

a **Deployment** is a higher-level concept that manages **ReplicaSets** and provides declarative updates to Pods along with a lot of other useful features. Therefore, we recommend using Deployments instead of directly using ReplicaSets

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
        app: guestbook
        tier: frontend
spec:
  # modify replicas according to your
case
  replicas: 3
  selector:
        matchLabels:
        tier: frontend
  template:
        metadata:
        labels:
        tier: frontend
        spec:
        containers:
        - name: php-redis
        image:
us-docker.pkg.dev/google-samples/
containers/gke/gb-frontend:v5
```

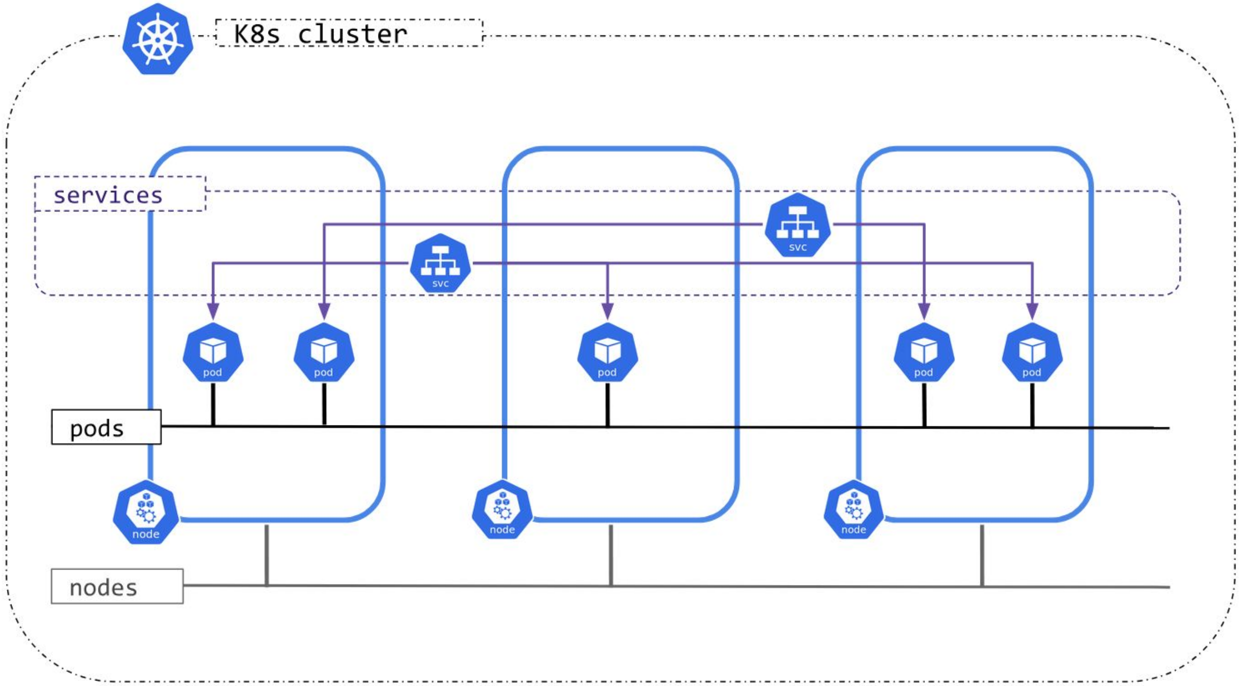
K8s Architecture – Networking

A pod has its own **private network namespace** which is shared by all of the containers within the pod

Each pod in a cluster gets its own unique **cluster-wide IP address.**

All pods can communicate with all other pods, whether they are on the same node or on different nodes.

Agents on a node (such as system daemons, or kubelet) can communicate with all pods on that node.



K8s Architecture – Services

Pods are ephemeral. When they terminate, so does their ip address.

Groups of pods can be given a permanent ip address called a **Service**

By default clients send requests to a stable internal IP address (**ClusterIP**)

Clients can also send requests to the IP address of a node and a **nodePort** specified by the Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-np-service
spec:
  selector:
    app: products
    department: sales
  type: NodePort
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

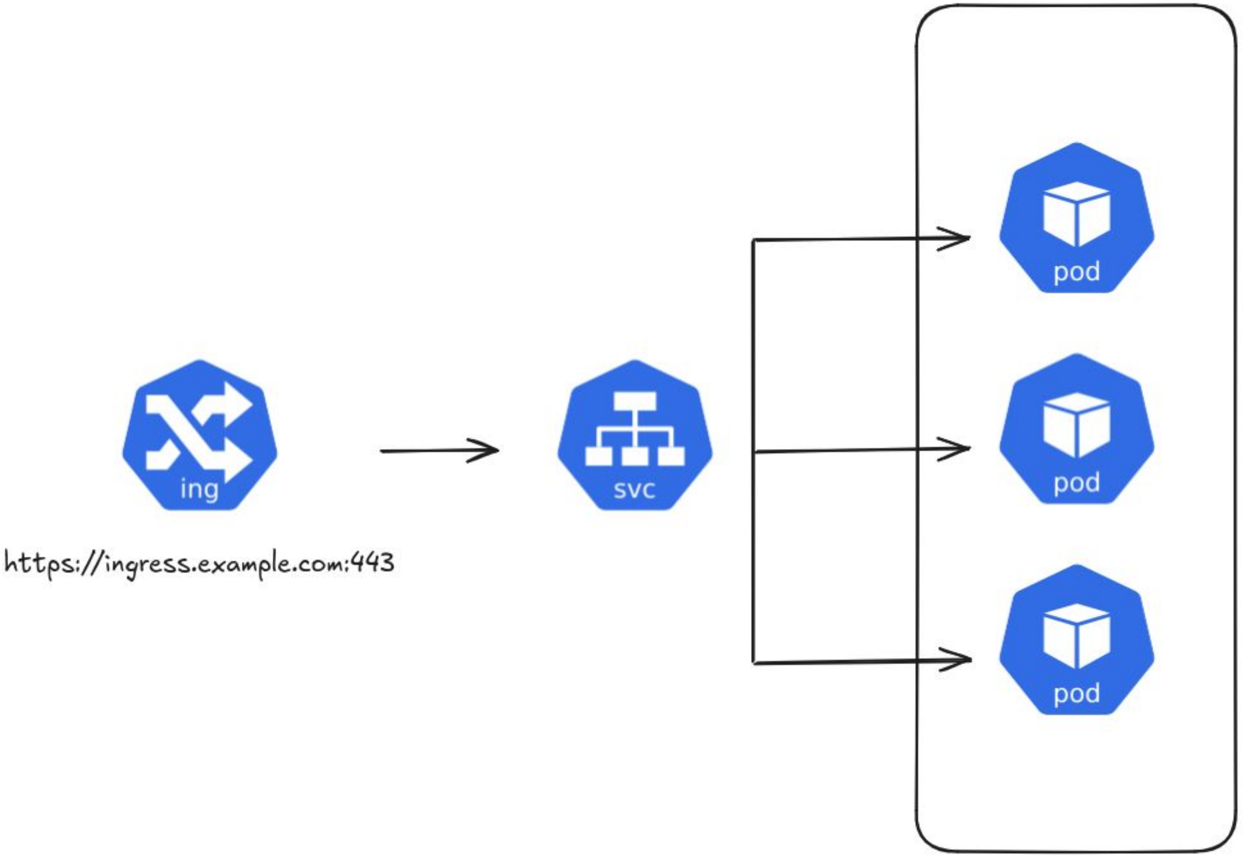
```
apiVersion: v1
kind: Service
metadata:
  name: my-cip-service
spec:
  selector:
    app: metrics
    department: sales
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

K8s Architecture – Ingress

An **Ingress** exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.

An Ingress may be configured to give Services externally-reachable URLs, load balance traffic, terminate TLS, and offer name-based virtual hosting.

An Ingress doesn’t expose arbitrary ports or protocols other than HTTP and HTTPS



K8s Architecture – ConfMap

A ConfigMap is used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: database-config
data:
  database_URL: "192.168.100.1/database"
  database_port: "3306"
```

K8s Architecture – Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code/container.

Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-sa-sample
  annotations:
        kubernetes.io/service-account.name:
"sa-name"
type: kubernetes.io/service-account-token
data:
  extra: YmFyCg==
```

Kubernetes Secrets are, by default, **stored unencrypted** in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret. In order to safely use Secrets, take at least the following steps: **(1) Enable Encryption at Rest for Secrets. (2) Consider using external Secret store providers.**

K8s Architecture – Volumes

Container files are temporary and lost when:

- Container crashes
- Container stops

This poses problems for applications that:

- Need persistent data
- Require consistent state

Solution: Use **K8s Volumes**, to retain data beyond container lifetimes.

```
apiVersion: v1

kind: PersistentVolume

metadata:

 name: foo-pv

spec:

 storageClassName: ""

 claimRef:

   name: foo-pvc

   namespace: foo

...
```

kubectl

Kubernetes provides a command line tool **kubectl** for communicating with a Kubernetes cluster's control plane, using the Kubernetes API.

Use the following syntax to run kubectl commands from your terminal window:

```
kubectl [command] [TYPE] [NAME] [flags]
```

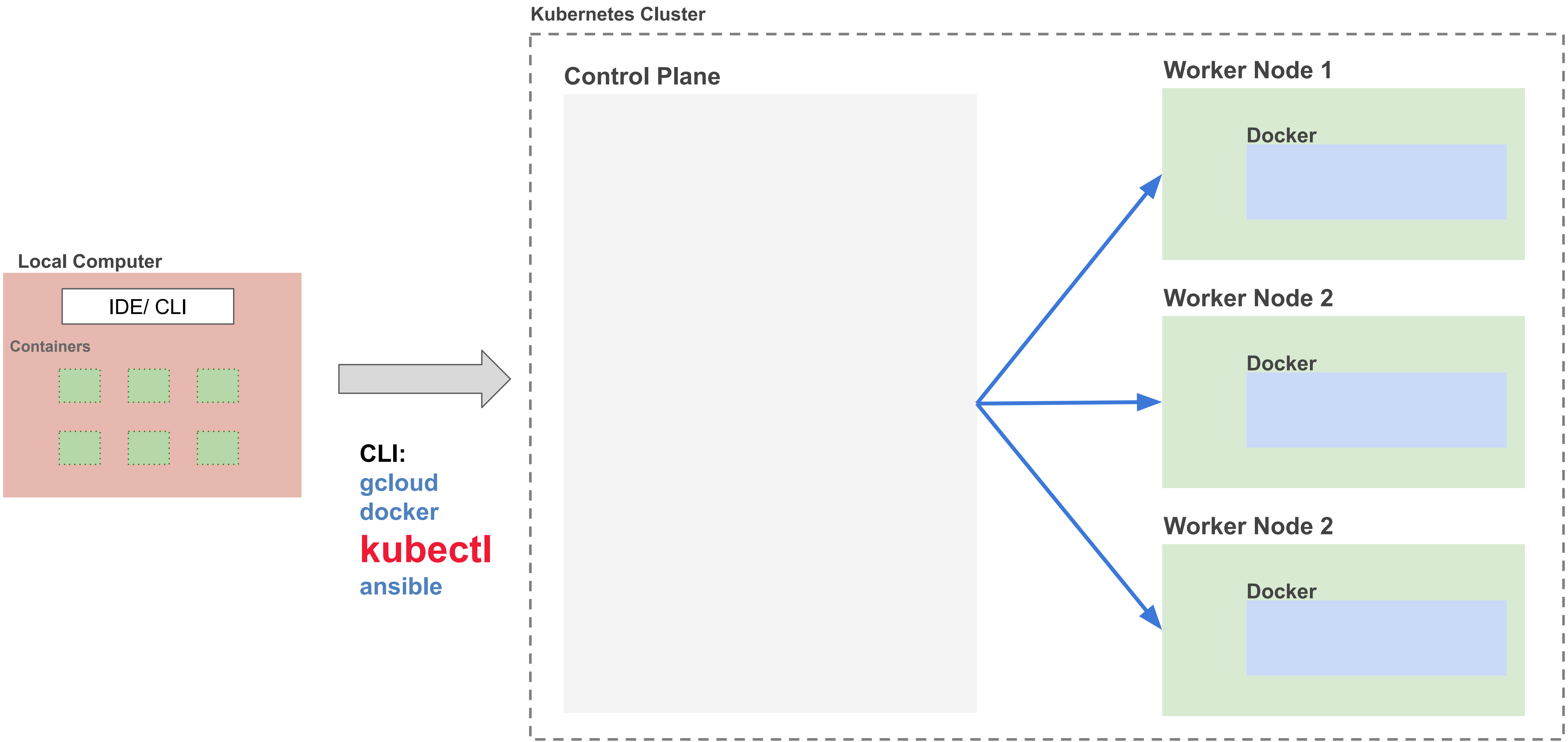
kubectl

```
# Create a service using the definition in example-service.yaml.
kubectl apply -f example-service.yaml

# Create a replication controller using the definition in example-controller.yaml.
kubectl apply -f example-controller.yaml

# Create the objects that are defined in any .yaml, .yml, or .json file within the <directory> directory.
kubectl apply -f <directory>
```

How do we build with Kubernetes?



Kubernetes Summary

- **Abstracting** Infrastructure
- **Standardize** Application Deployment
- Deploy Applications **Declaratively**
- Daily **Management** of Applications

Outline

1. Recap
2. Motivation
3. Introduction to Kubernetes
4. **Tutorial: Deploying a Kubernetes Cluster**
5. Advantages of using Kubernetes

Create Kubernetes Cluster

To create a Kubernetes cluster

- You must first install ***gcloud*** which is the GCPs command-line tool
- You create and delete clusters using ***gcloud***

Example:

Create a 2 node Kubernetes Cluster

```
gcloud container clusters create test-cluster --num-nodes 2 --zone us-east1-c
```

```
Creating cluster test-cluster in us-east1-c...⠛
```

Create Kubernetes Cluster

Create a 2 node Kubernetes Cluster

```
gcloud container clusters create test-cluster --num-nodes 2 --zone us-east1-c
```

```
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/...
kubeconfig entry generated for test-cluster.
NAME          LOCATION    MASTER_VERSION  MASTER_IP      MACHINE_TYPE  NODE_VERSION    NUM_NODES  STATUS
test-cluster  us-east1-c  1.20.9-gke.701  34.73.126.138  e2-medium     1.20.9-gke.701  2          RUNNING
```

Deploying to Kubernetes Cluster

To create a Kubernetes cluster and deploy app to it.

- You must first install ***kubectl*** which is the Kubernetes command-line tool
- You can manage all resources in Kubernetes using ***kubectl***

Examples:

Get version of client

```
kubectl version --client
```

```
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.1",
GitCommit:"632ed300f2c34f6d6d15ca4cef3d3c7073412212",
GitTreeState:"clean", BuildDate:"2021-08-19T15:45:37Z",
GoVersion:"go1.16.7", Compiler:"gc", Platform:"linux/amd64"}
```

Get version of server

```
kubectl version
```

```
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.1",
GitCommit:"632ed300f2c34f6d6d15ca4cef3d3c7073412212",
GitTreeState:"clean", BuildDate:"2021-08-19T15:45:37Z",
GoVersion:"go1.16.7", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you
specify the right host or port?
```

Deploying to Kubernetes Cluster

Examples:

Get Kubernetes Cluster Information

```
kubectl get all
```

```
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.3.240.1   <none>        443/TCP   48m
```

Get Kubernetes Component Status

```
kubectl get componentstatuses
```

```
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
etcd-1               Healthy   {"health":"true"}
controller-manager   Healthy   ok
etcd-0               Healthy   {"health":"true"}
```

Deploying to Kubernetes Cluster

Examples:

Get Kubernetes Cluster Nodes

```
kubectl get nodes
```

```
NAME                                          STATUS   ROLES    AGE   VERSION
gke-test-cluster-default-pool-2e9eafc9-kj0s   Ready    <none>   51m   v1.20.9-gke.701
gke-test-cluster-default-pool-2e9eafc9-t4pw   Ready    <none>   51m   v1.20.9-gke.701
```

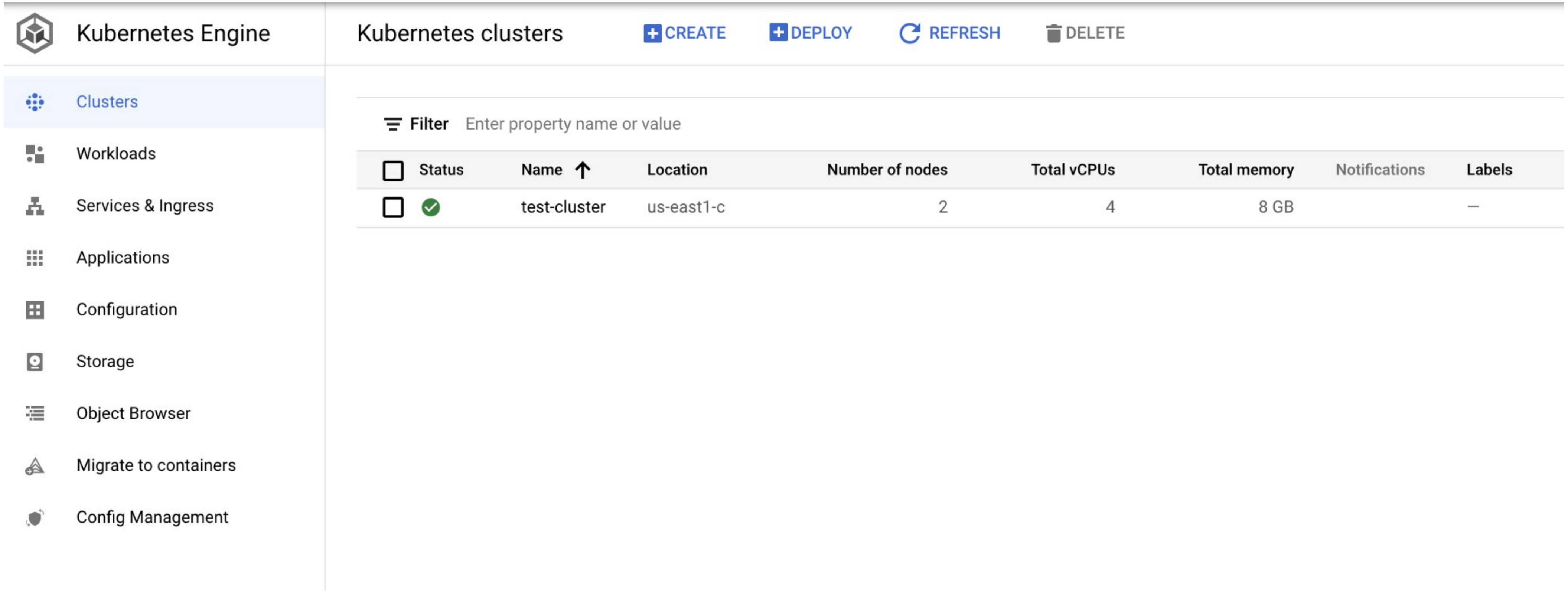
Get Kubernetes Pods

```
kubectl get pods
```

```
No resources found in default namespace.
```

Deploying to Kubernetes Cluster

You can view Kubernetes cluster details directly from GCP



□	Status	Name ↑	Location	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
□	✓	test-cluster	us-east1-c	2	4	8 GB		—

Deploying to Kubernetes Cluster

Examples:

Deploy App to Kubernetes

```
kubectl apply -f deploy-k8s-tic-tac-toe.yml
```

```
deployment.apps/web created
service/web created
```

Get Services

```
kubectl get services
```

```
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.3.240.1    <none>           443/TCP        29m
web          LoadBalancer   10.3.242.77   34.139.195.206   80:32088/TCP   3m51s
```

Deploying to Kubernetes Cluster

Deployment YAML

```
---
apiVersion: apps/v1
kind: Deployment
spec:
  replicas: 2
  containers:
   - image: dlops/tic-tac-toe
     imagePullPolicy: IfNotPresent
     name: web
     ports:
      - containerPort: 8080
        protocol: TCP
```

Deployment:

- Decares what is in a pod and how many replicas
- Is in charge of keeping the pod running

Service YAML

```
---
apiVersion: v1
kind: Service
spec:
 ports:
 - port: 80
   protocol: TCP
   targetPort: 8080
 type: LoadBalancer
```

Service:

- Decares how traffic is routed to a pod or a multiple replicas.
- Service allows pods to die

Deleting a Kubernetes Cluster

Example:

Delete Kubernetes Cluster called test-cluster

```
gcloud container clusters delete test-cluster --zone us-east1-c
```

```
The following clusters will be deleted.
 - [test-cluster] in [us-east1-c]

Do you want to continue (Y/n)?  Y

Deleting cluster test-cluster...done.
Deleted [https://container.googleapis.com/v1/projects/.../zones/us-east1-c/clusters/test-cluster].
```

Tutorial: Deploying a Kubernetes Cluster

Deploying a Kubernetes Cluster

Run an ansible playbook now for the cheese app.

https://github.com/dlops-io/cheese-app-v3?tab=readme-ov-file#deployment-with-scaling-using-kubernetes

Outline

1. Recap
2. Motivation
3. Introduction to Kubernetes
4. Tutorial: Deploying a Kubernetes Cluster
5. **Advantages of using Kubernetes**

Advantages of using Kubernetes

- **Self-Service** Deployment of Applications
- **Reduce Cost** by better Infrastructure Utilization
- **Automatically Adjusting** to varying loads
- Running Applications **Smoothly**
- Simplifying Application **Development**

THANK YOU