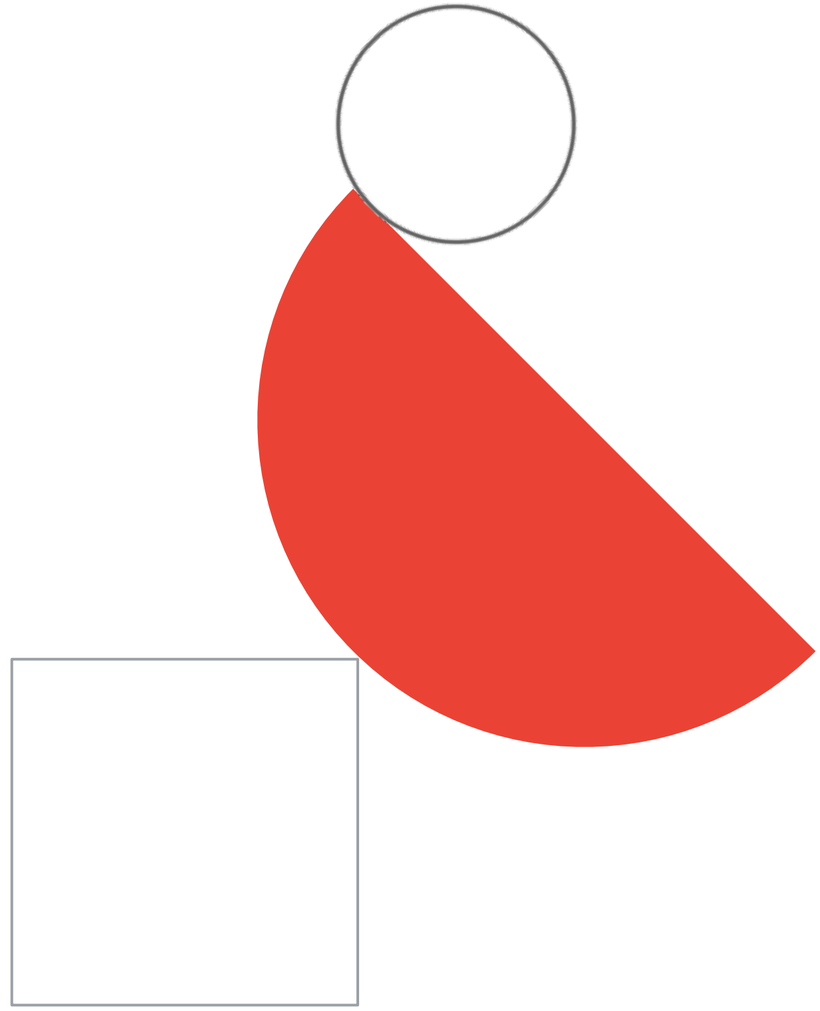
Google Kubernetes Engine (GKE)



GKE: Autopilot Mode

GKE manages underlying infrastructure of the cluster, including the nodes

High availability

Regional cluster; Regular Release Channel; Auto-Update; Auto-Repair; Surge Upgrade;

Network

VPC Native (alias IP); IP-friendly (limit cluster size/ pods per node); full network flexibility

Highly Scalable

Node Auto Provision; Horizontal Pod Autoscaler; Vertical Pod Autoscaler

Secured by default

Workload Identity; Shielded Nodes; Secure-boot-disk; COS and Containerd, block known unsecure features.

Pod Disruption Budget, Readiness and Liveness Probes

A **PDB (Pod Disruption Budget)** limits the number of pods of a replicated application that can be taken down **simultaneously** from **voluntary** disruptions.

An Application Owner can create a **PodDisruptionBudget** object (PDB) for each application.

Readiness probes: designed to know when your app is ready to serve traffic.

Liveness probes: designed to let Kubernetes know if your app is alive or dead.

```
kind: PodDisruptionBudget
metadata:
 name: km-pdb
spec:
 minAvailable: 2
 selector:
   matchLabels:
     app: kobimysql
```

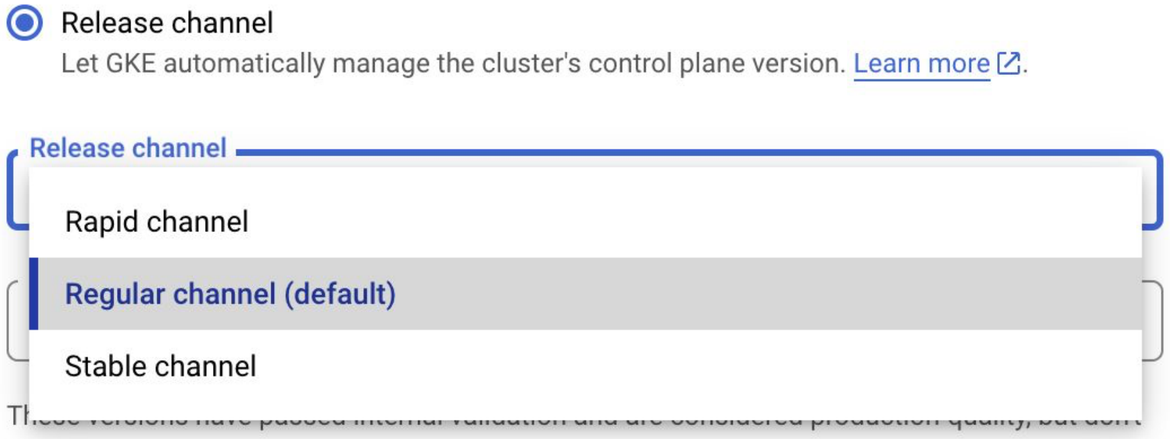
```
maxUnavailable: 1
```

Exam Tip:

- *See how to ensure stateful workloads are disruption-ready*
- *Great explanation of Readiness and Liveness probes here.*

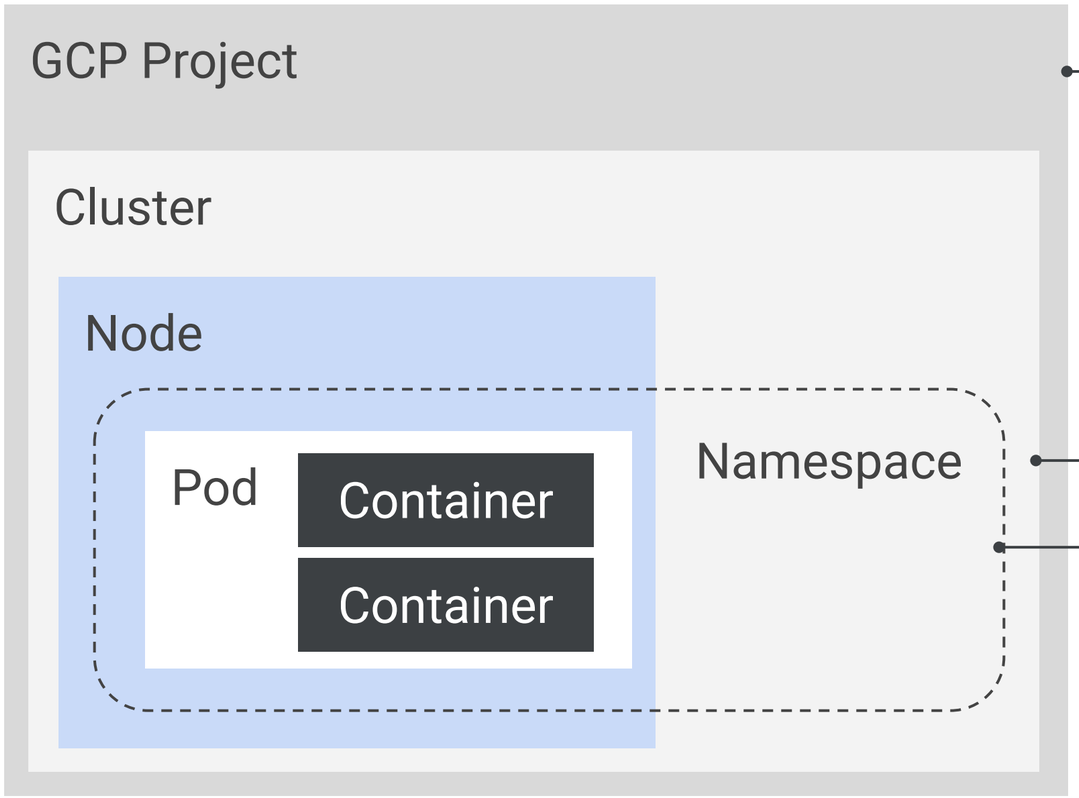
Best practices for GKE upgrades

1. **Setup multiple environments:** at a minimum pre-production and production clusters
2. **Enroll Clusters in Release Channels**: Stable or Regular release channels for production clusters.



3. **Create continuous upgrade strategy**: Receive updates about new GKE versions through cluster upgrade notifications through Pub/Sub
4. **Schedule maintenance windows and exclusions**: to increase upgrade predictability
5. **Set tolerance for disruption**: To ensure that pods sufficient number of replicas, use Pod Disruption Budget

GKE: Using IAM and RBAC



Use IAM at the project level
Set roles for

- Cluster Admin: manage clusters
- Container Developer: API access within clusters

Use RBAC at the cluster and namespace level
Set permissions on individual clusters and namespaces

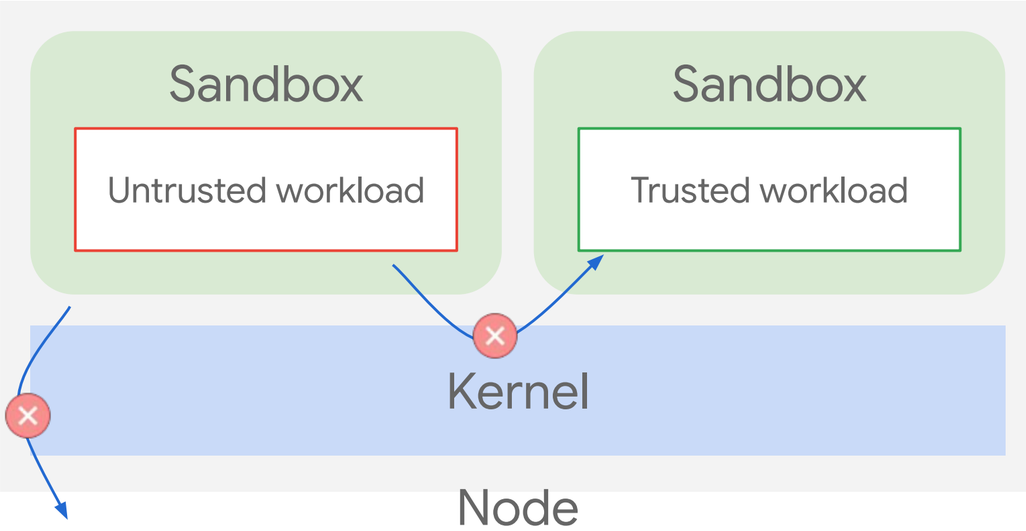
***Exam Tip**: IAM and Kubernetes RBAC work together to help manage access to your cluster. RBAC controls access on a cluster and namespace level, while IAM works on the project level*

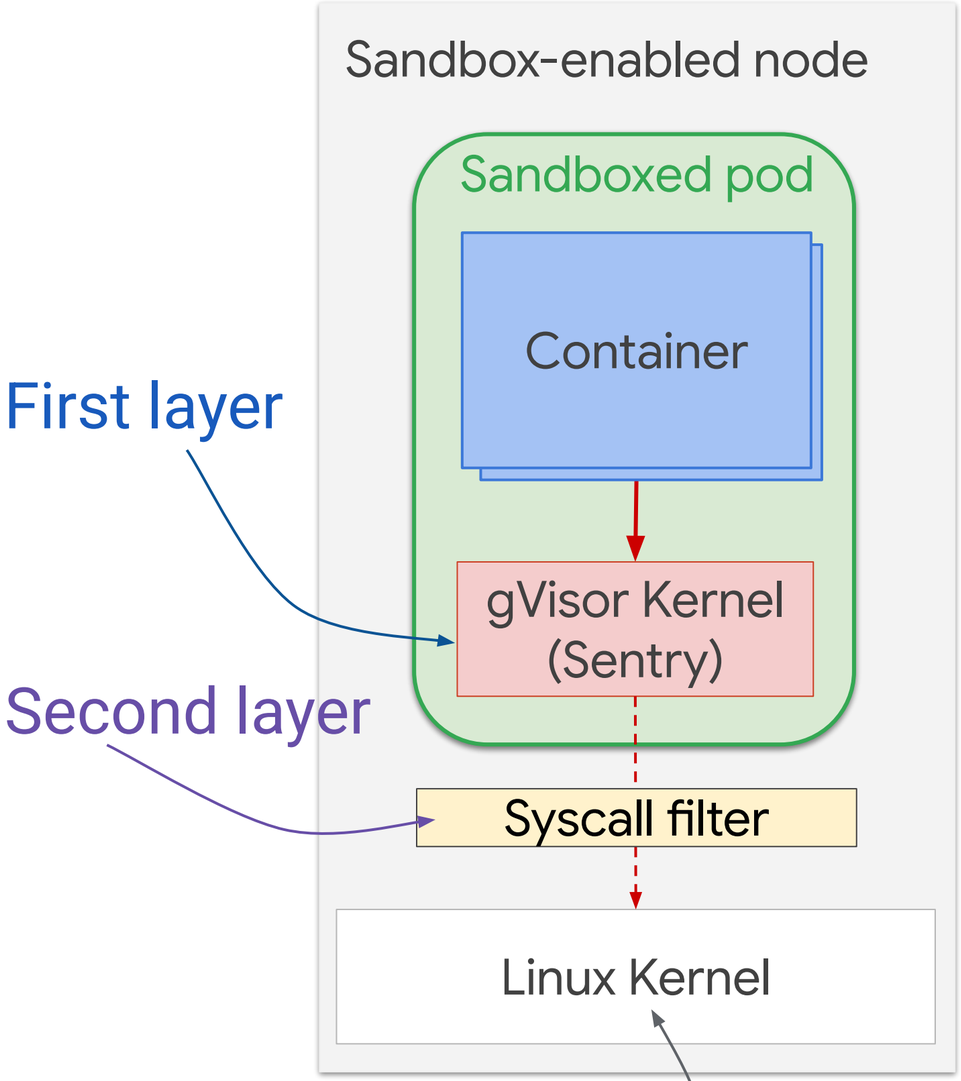
GKE Sandbox

Run **trusted and untrusted** workloads on the same node

Rather than achieving isolation via separate VMs, you can run workloads of different trust levels on the same node

Performance improvements from not having to allocate a new cluster to achieve isolation





***Exam Tip**: Commit 10 minutes to get an overview ofGKE Sandbox*

GKE networking: Subnet sizes

Subnet size for nodes	Maximum nodes	Maximum Pod IP addresses needed	Recommended Pod address range
/29	4	1,024	/21
/28	12	3,072	/20
/27	28	7,168	/19
/26	60	15,360	/18
/25	124	31,744	/17
/24	252	64,512	/16
/23	508	130,048	/15
/22	1,020	261,120	/14
/21	2,044	523,264	/13
/20	4,092	1,047,552	/12
/19	8,188	2,096,128	/11 (maximum Pod address range)

***Exam Tip**: make sure to watch this video to understand GKE networking well!*

GKE networking: Example

Subnet size for nodes	Maximum nodes	Maximum Pod IP addresses needed	Recommended Pod address range
/29	4	1,024	/21

$2^{(32-29)} = 8$ (4 of these are reserved for GCP)

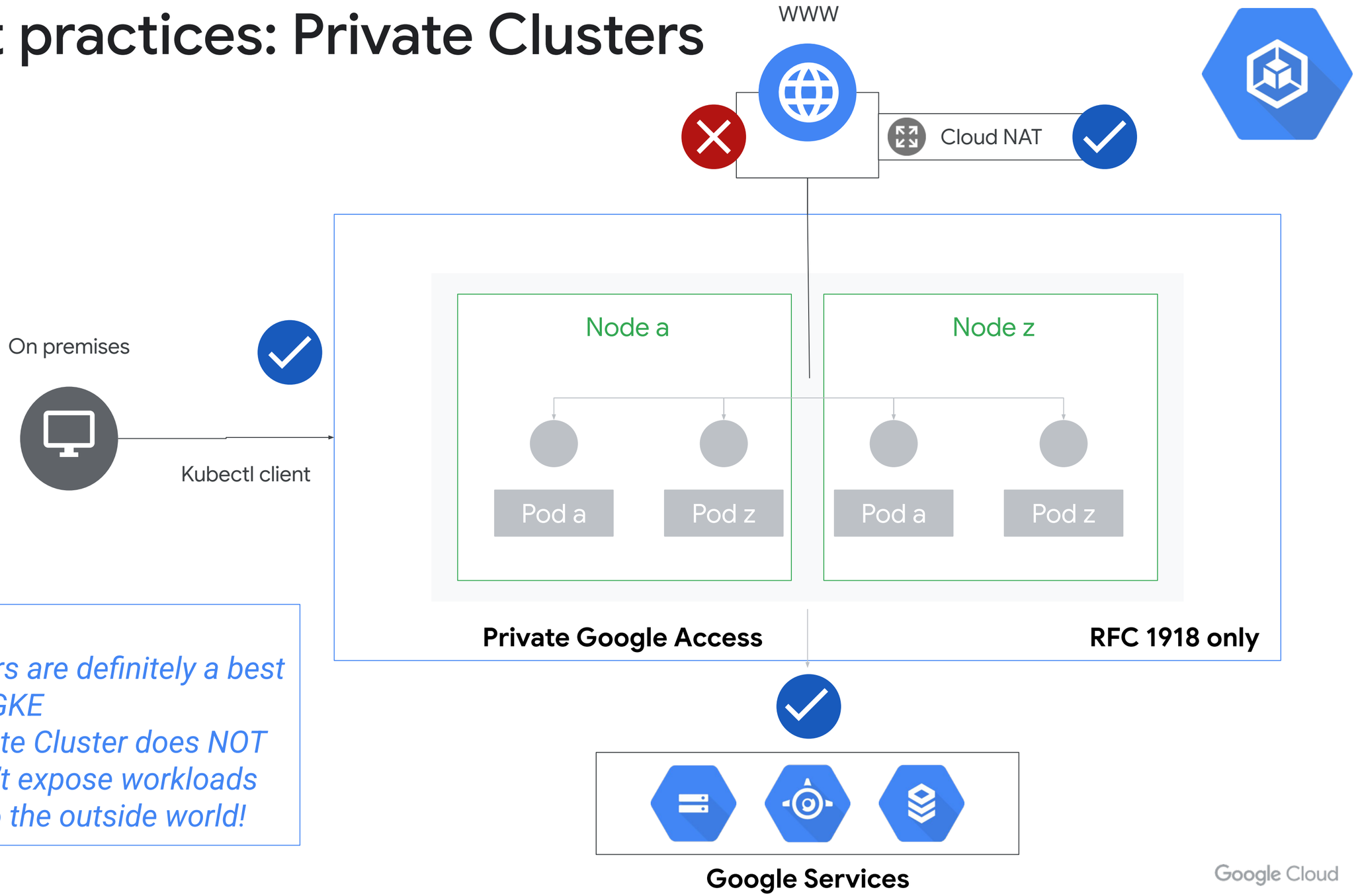
110 pods running in each node -> **4 * 110 = 440**

Twice number of IPs per pod **440 * 2 = 880**

2^{10} number of IPs for pods

*Assuming the default maximum of 110 pods per node

GKE best practices: Private Clusters



Exam Tip:

- *Private Clusters are definitely a best practice with GKE*
- *Having a Private Cluster does NOT mean you can't expose workloads via Services to the outside world!*

Container best practices: building images

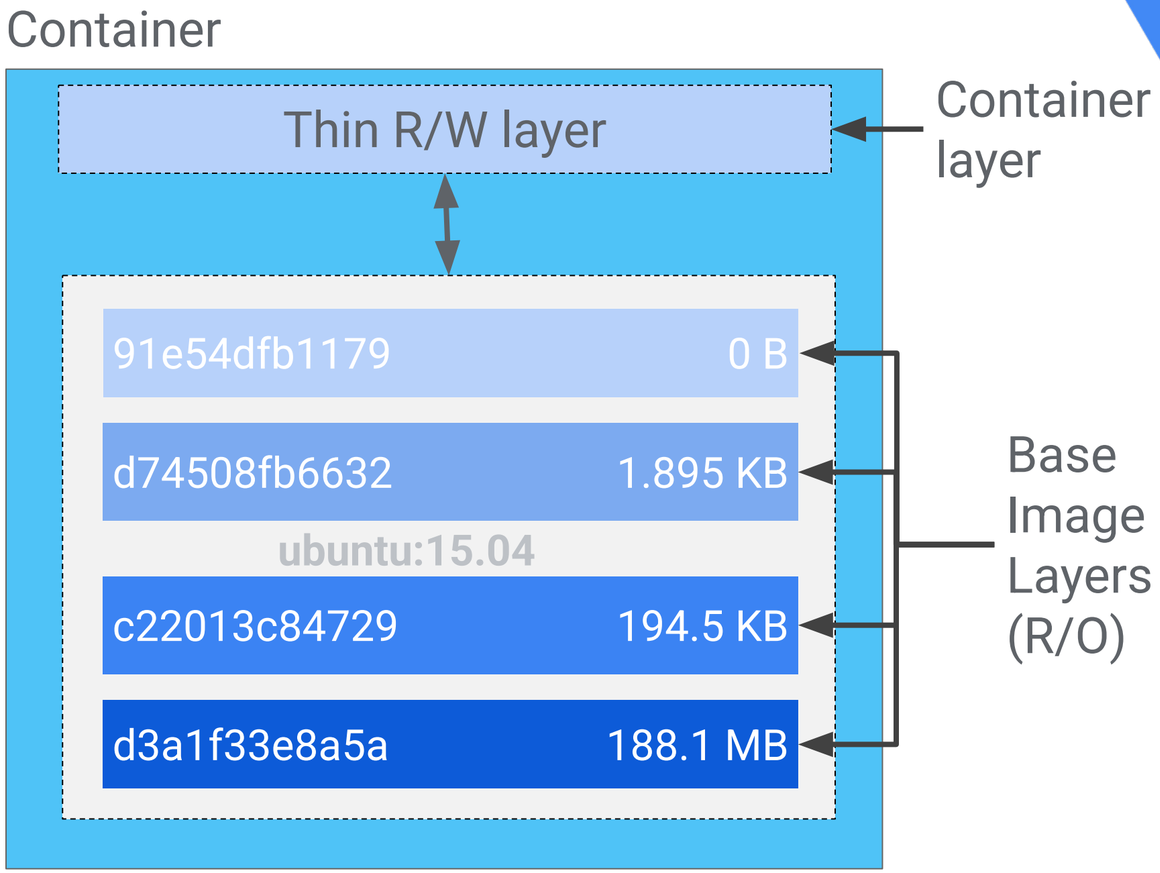
Exam Tip: *Each instruction in a Dockerfile creates a layer.*

Dockerfile

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

docker commands

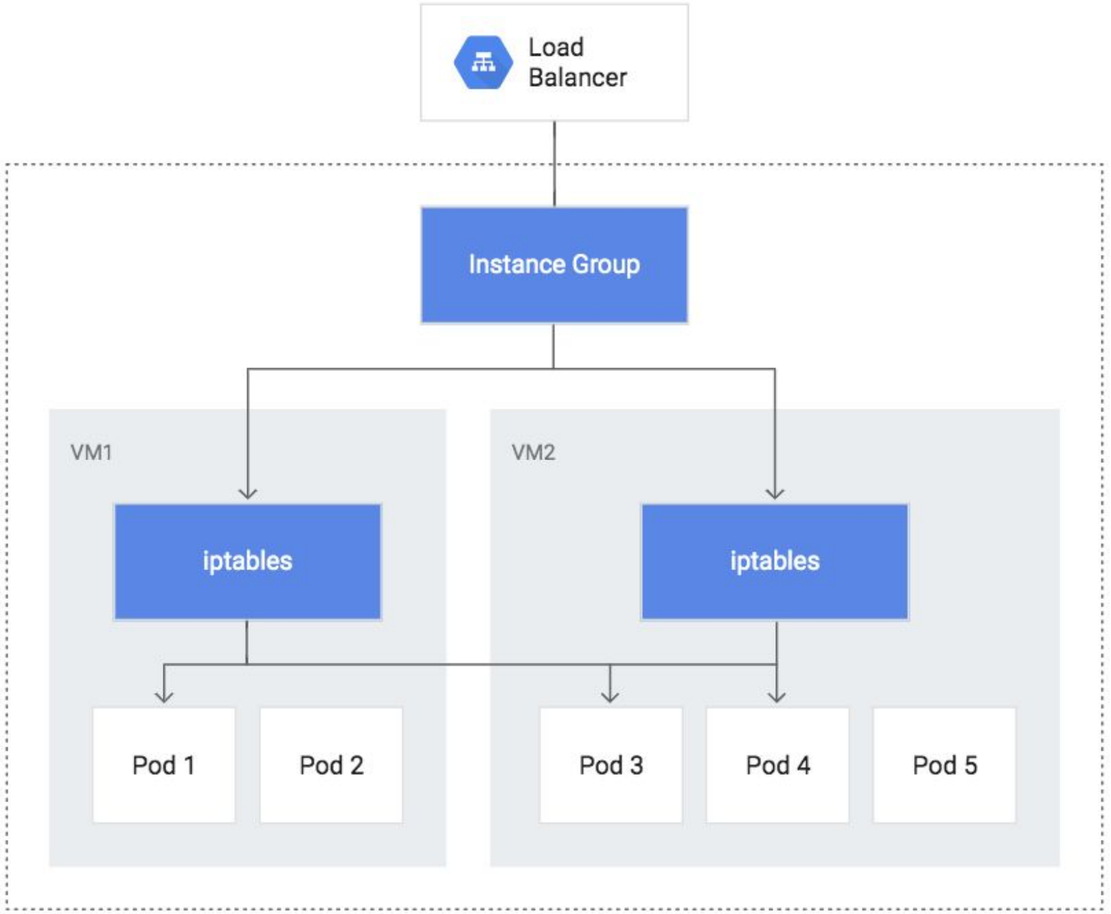
```
$> docker build -t py-web-server
.
$> docker run -d py-web-server
$> docker images
$> docker ps
$> docker logs <container id>
```

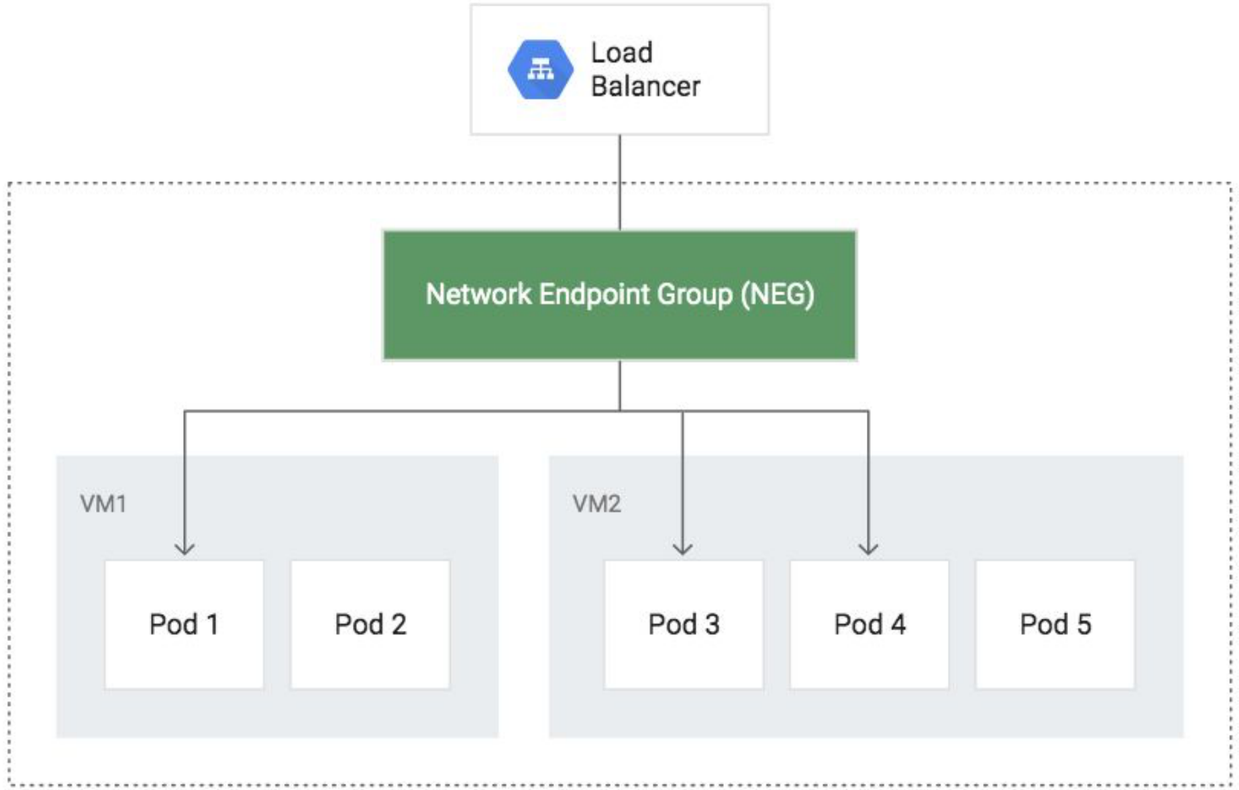


Exam Tips: *Here are best practices for building container images:*

- *Use the smallest base image possible (when new versions are rolled out, only smallest image layers are changed). Eg. use "alpine" image rather than "centos" or "ubuntu" if possible.*
- *Use multi-stage builds (app can be built in a first "build" container and the result can be used in another container)*
- *Try to create images with common layers (if a layer already exists on a cluster, it does not have to be downloaded)*

Ingress service: standard (non-NEG) vs NEG





Exam Tip: *NEG is often preferred as a container-native load balancing type.*

GCP API access from k8s *without* Workload Identity

[Authenticate to Google Cloud using a service account | Kubernetes Engine]

- Create a GCP Service Account (GSA)
- Create Keys for GSA
- Import GSA Keys as a k8s Secret
- For the k8s Workload:
  - Define a Volume with the Secret
  - Mount the Volume inside the container
  - Point $GOOGLE_APPLICATION_CREDENTIALS at the key file
- Workload can now authenticate to GCP APIs as the GSA

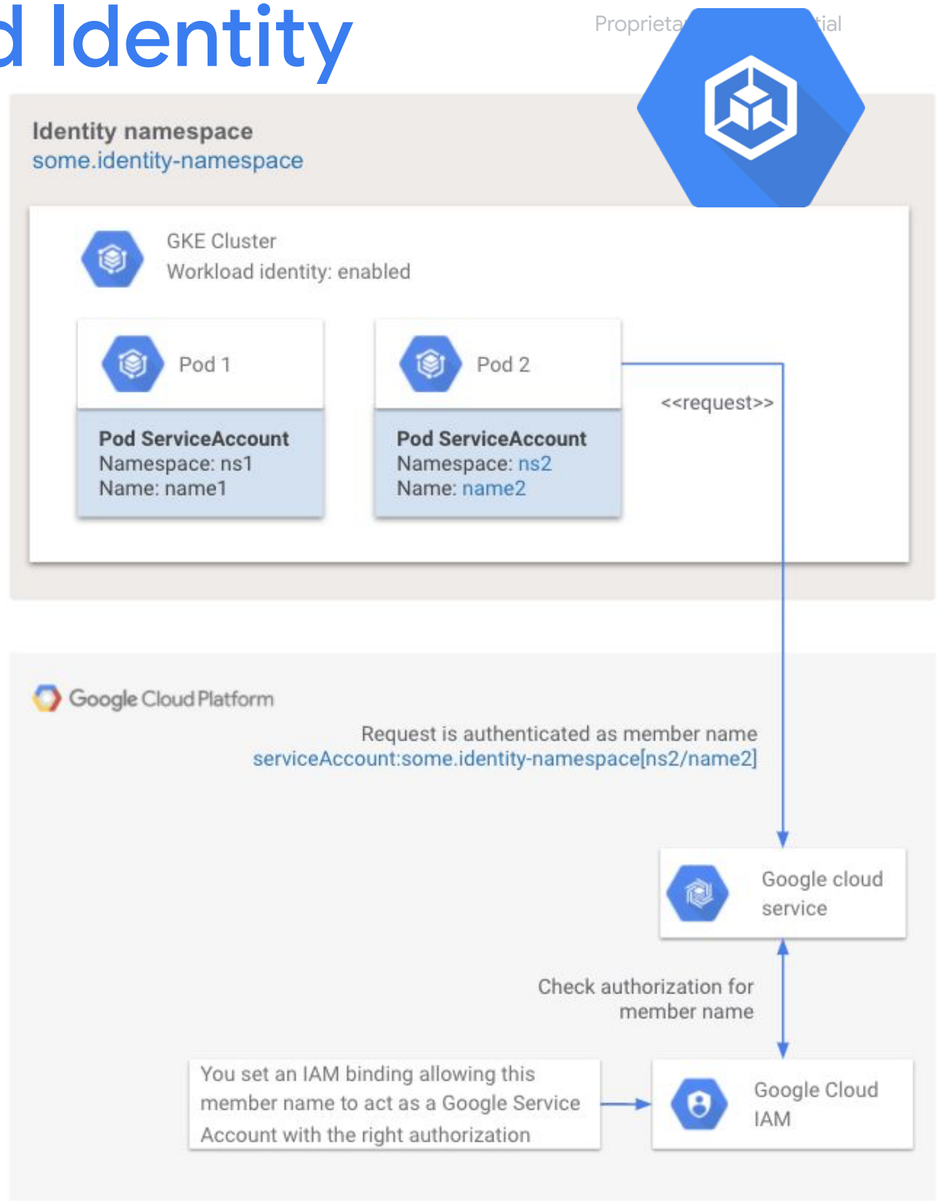
=> toilsome to setup & hard to secure

GKE API access from k8s *with* Workload Identity

- Enable Workload Identity for the GKE cluster
- Run workload using a dedicated k8s service account (KSA)
- Grant KSA access to desired GCP resources using IAM roles
- Workload can now access GCP APIs by presenting (short-lived, auto-rotated) KSA tokens

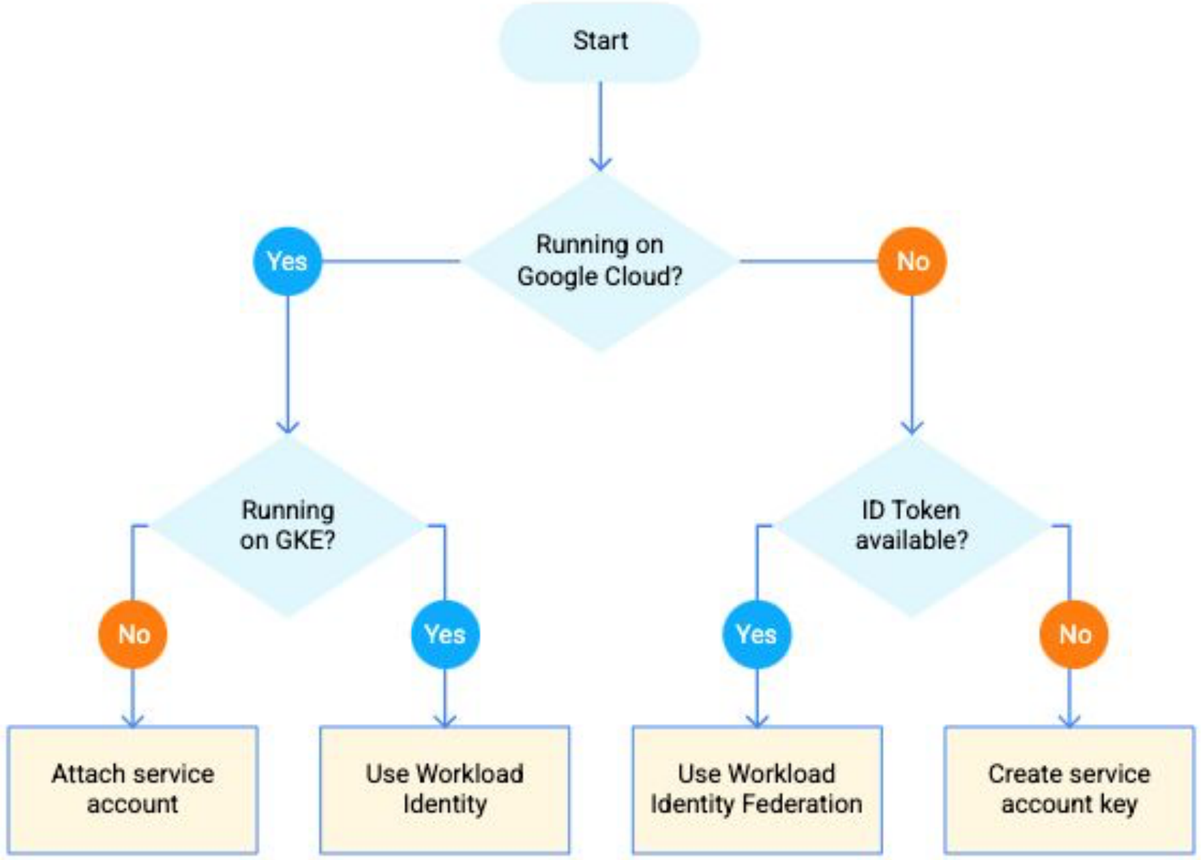
It just works!

Exam Tip: *Workload Identity is a best practice for a GKE which needs to access other GCP APIs.*



Workload Identity vs Workload Identity Federation

- Those are two different things! Both aim at limiting usage of Service Account keys, but:
  - Workload Identity = used when microservices deployed to your GKE cluster need to access other GCP resources / APIs.
  - Workload Identity Federation = when some services of yours deployed outside of GCP (in on-premises or other hyperscalers) need to access GCP resources / APIs.

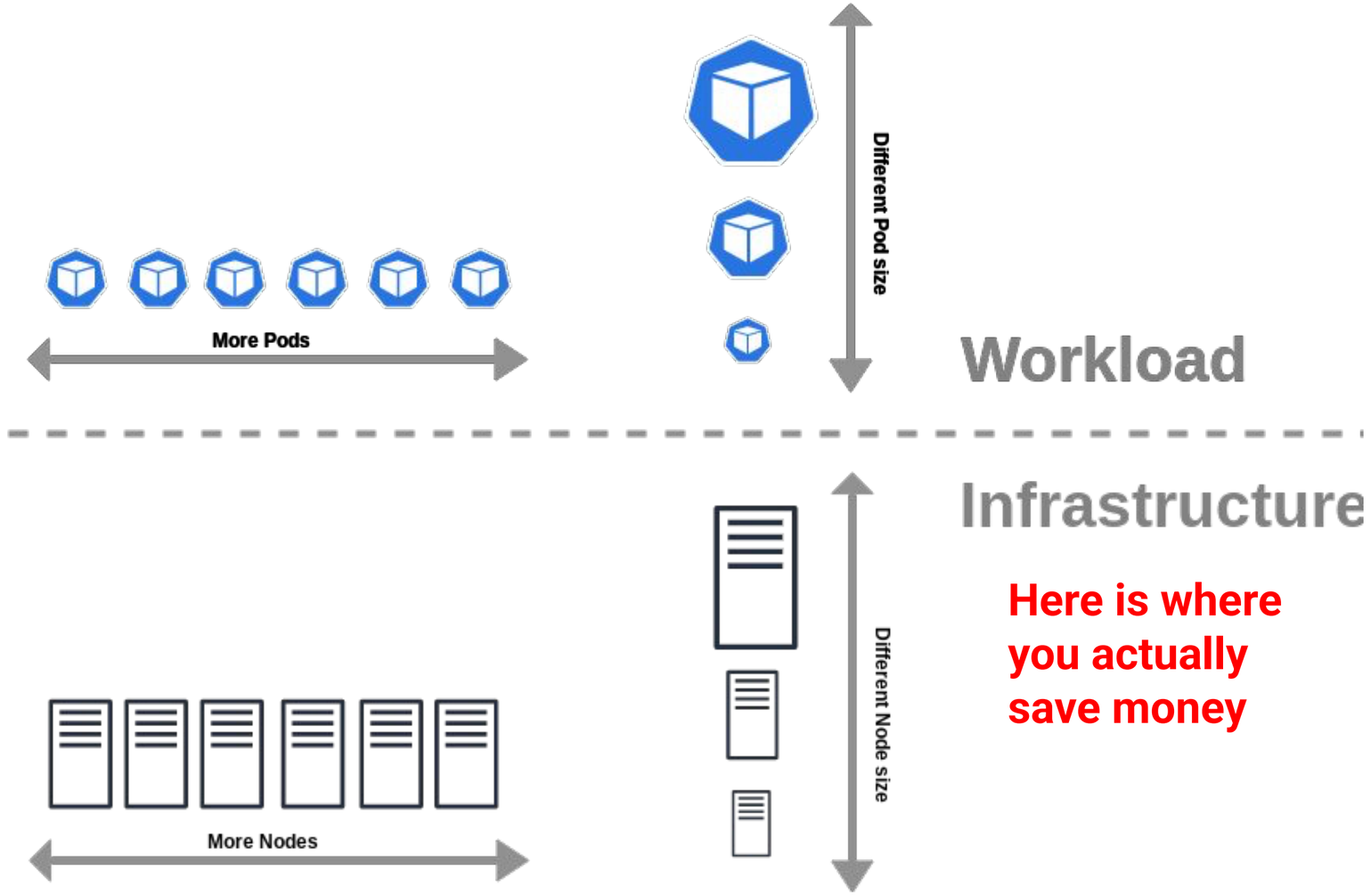


How to bootstrap / change a GKE cluster

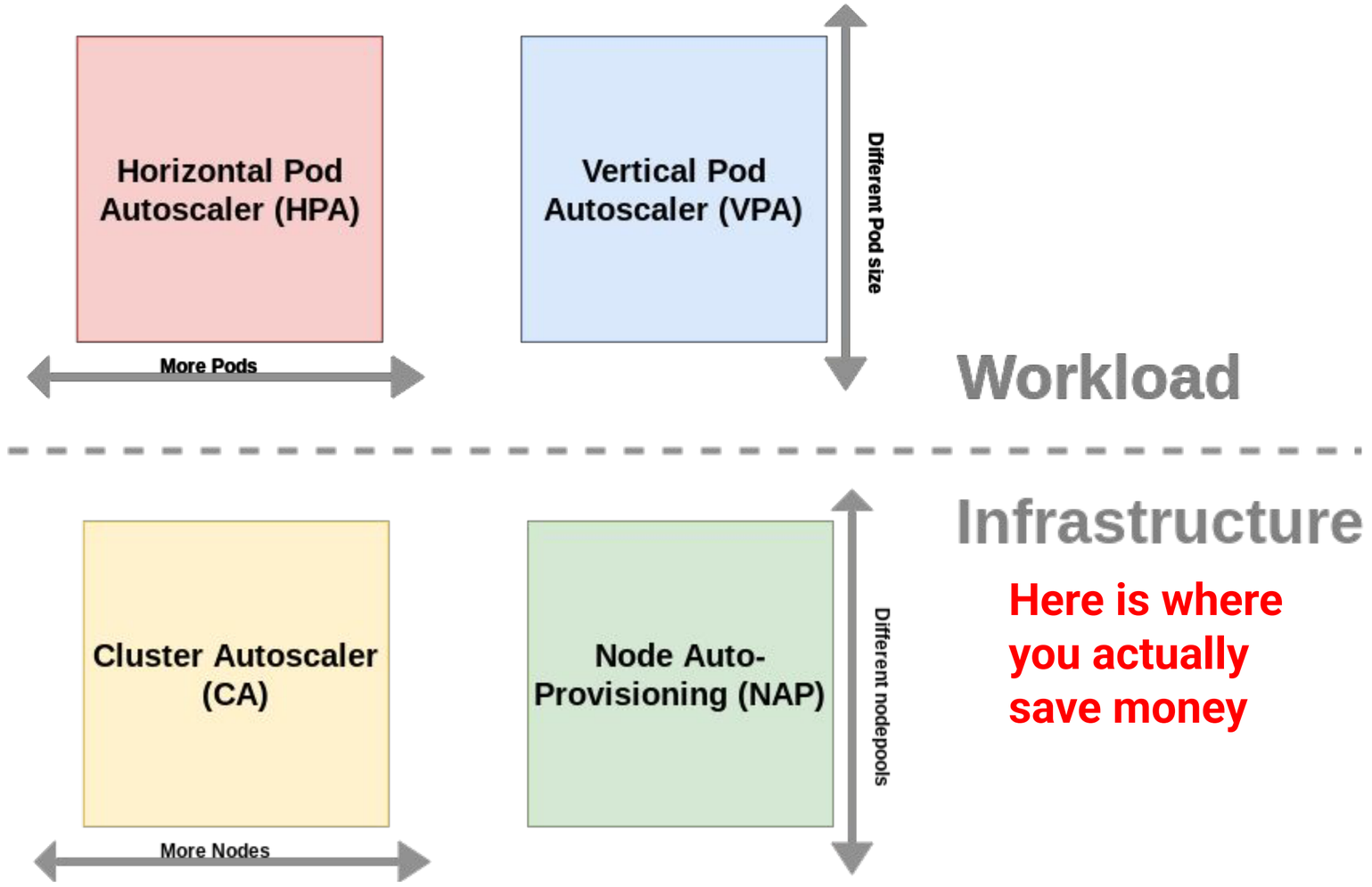
Exam Tips:

- *Make sure to differentiate:*
  - *When creating / modifying / deleting cluster (or its node pools), you should use gcloud command (since you're interacting with GCP to manage INFRASTRUCTURE for GKE). For example:*
    - *To create a cluster: 'gcloud container clusters create... '*
    - *To resize a cluster (change number of nodes): 'gcloud container clusters **resize** CLUSTER_NAME --node-pool POOL_NAME --num-nodes NUM_NODES'*
    - *To enable autoscaling on a node pool of existing cluster: 'gcloud container clusters **update** CLUSTER_NAME **--enable-autoscaling** --node-pool=POOL_NAME --min-nodes=MIN_NODES --max-nodes=MAX_NODES --region=COMPUTE_REGION'*
    - *To disable autoscaling on a node pool of existing cluster: 'gcloud container clusters update CLUSTER_NAME **--no-enable-autoscaling** --node-pool=POOL_NAME --region=COMPUTE_REGION'*
  - *When interacting with Kubernetes objects (eg. you'd like to deploy some Pods), you should use 'kubectl' command, eg:*

GKE: The 4 scalability dimensions

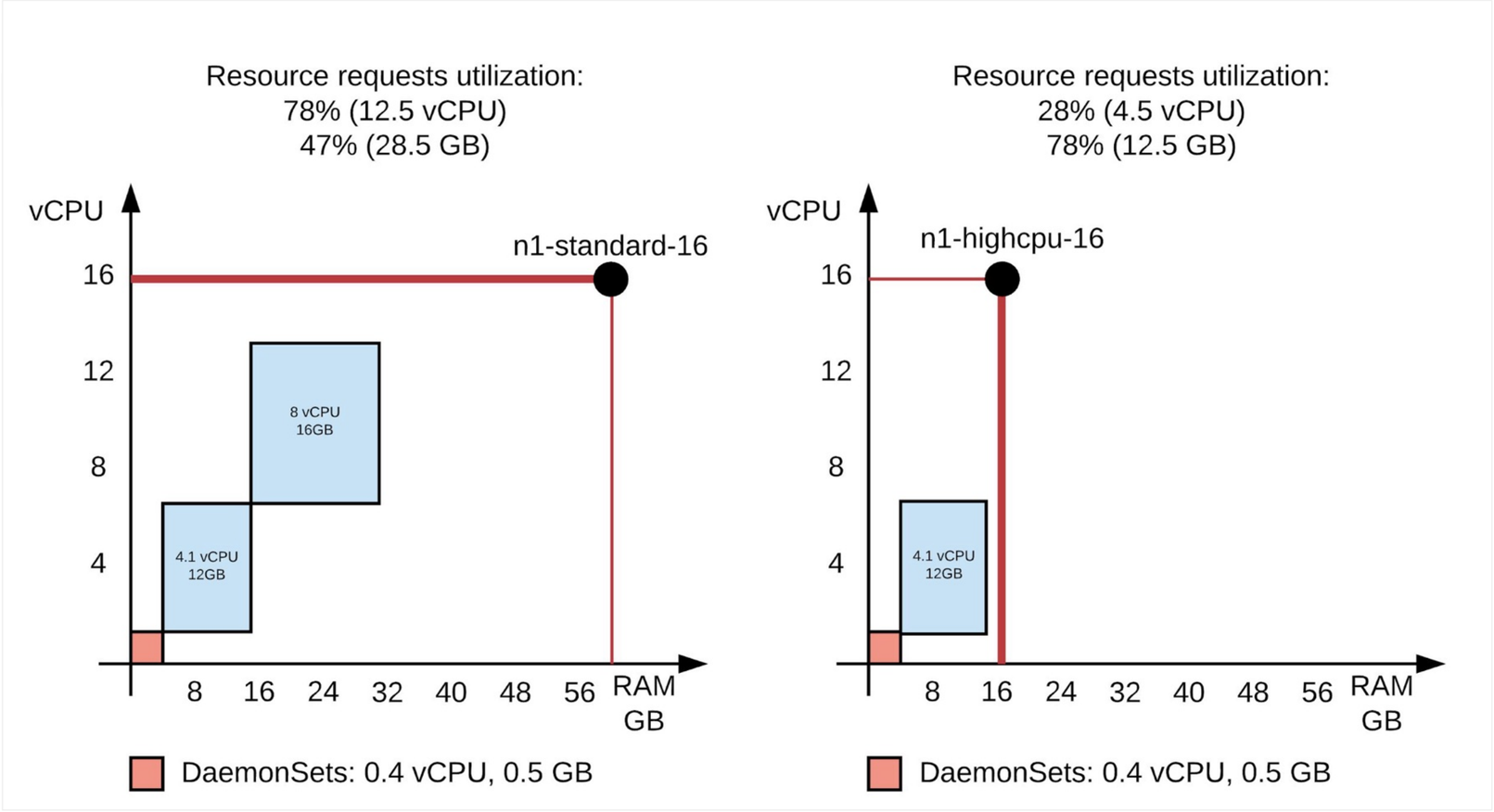


GKE supports all 4 scalability dimensions



GKE: Binpacking

- Make sure your workload fit well inside the machine size
- You can create multiple node pools and use either nodeSelector or Node Affinity to select which node your pod must run.
- Another simpler option is to configure Node auto-provisioning



GKE: Pod Placement

Requests & Limits	**Requests** specify how much resource (i.e. CPU and memory) a Container needs **Limits** specify the amount of resources the container is allowed to use
Node Selector	**Node selector** is an approach to schedule Pods to a specific set of nodes (or GKE node pools) using matching labels
Affinity & Anti-Affinity	**Affinity/anti-affinity i**s a scheduling feature to place Pods to Nodes using expressive rules against Pod and Node labels.
Taints & Tolerations	**Taints** are used to repel Pods from specific Nodes. **Tolerations** allow Pods to tolerate the taints

Pod Placement

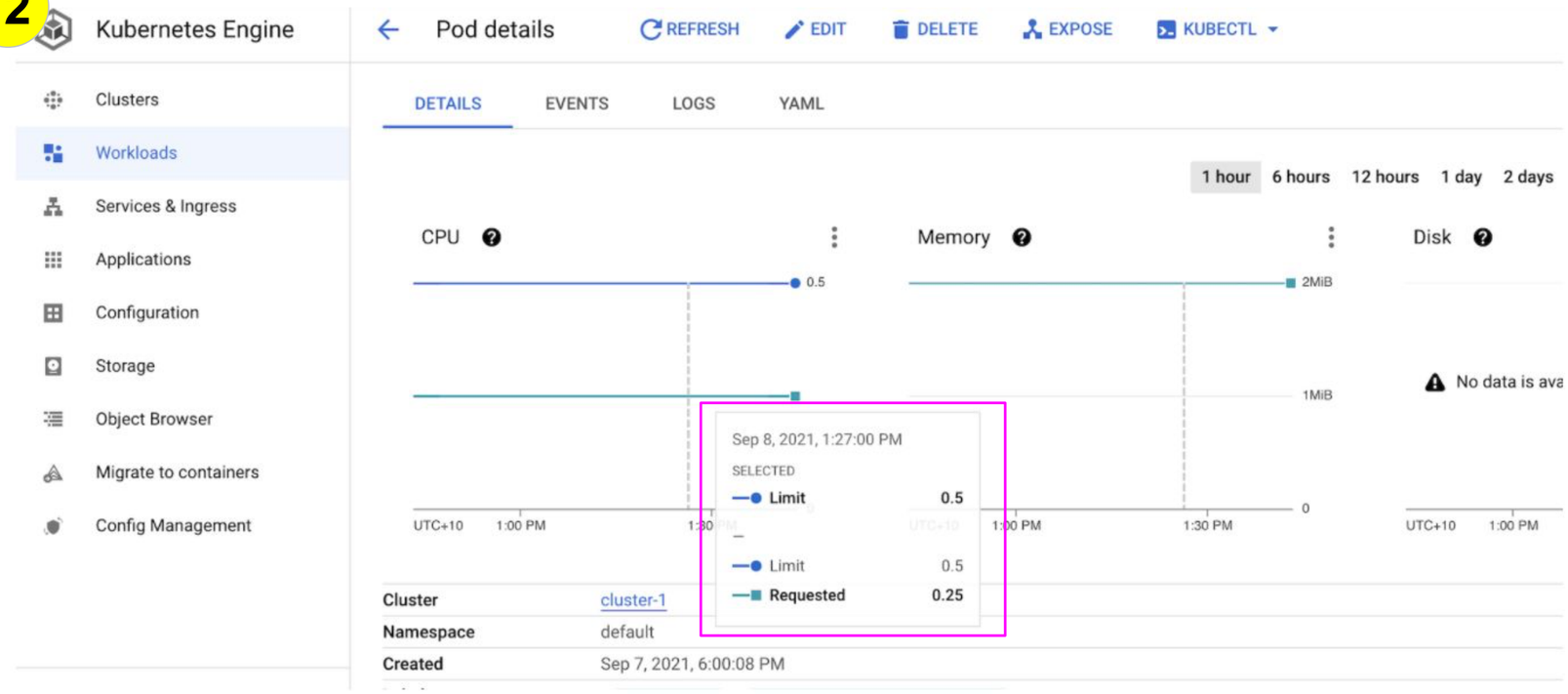
Requests & Limits

Requests specify how much resource (i.e. CPU and memory) a Container needs
Limits specify the amount of resources the container is allowed to use

1

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

2



Memory cgroup out of memory: Killed process - when Container hits a memory limit

Pod Placement

Node Selector

Node selector is an approach to schedule Pods to a specific set of nodes (or GKE node pools) using matching labels

1

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx:latest
    imagePullPolicy: IfNotPresent
  nodeSelector:
    cloud.google.com/gke-nodepool: app-pool
```

2

```
apiVersion: v1
kind: Pod
metadata:
  name: podthree
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx:latest
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

Pod Placement

Affinity & Anti-Affinity

Affinity/anti-affinity is a scheduling feature to place Pods to Nodes using expressive rules against Pod and Node labels.

```
pima@cloudshell:~/yaml-sample (first-medium-328400)$ cat redis.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  selector:
    matchLabels:
      app: redis
  replicas: 3
  template:
    metadata:
      labels:
        app: redis
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - redis
            topologyKey: "kubernetes.io/hostname"
      containers:
      - name: redis-server
        image: redis:latest
pima@cloudshell:~/yaml-sample (first-medium-328400)$
```

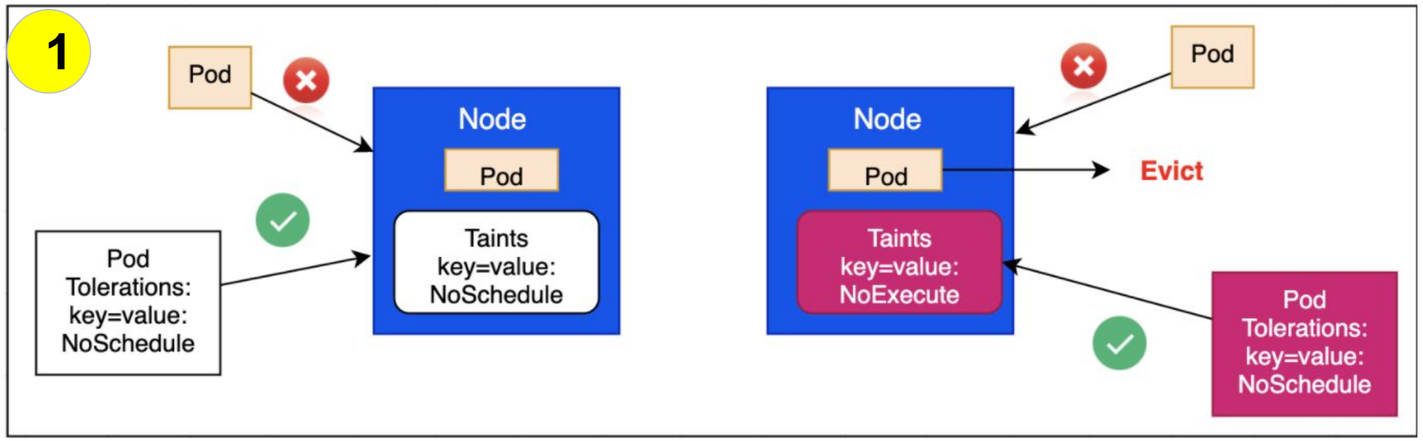
```
Error: Cannot schedule pods: node(s) didn't
match pod anti-affinity rules
```

Pod Placement

Taints & Tolerations

Taints are used to repel Pods from specific Nodes.
Tolerations allow Pods to tolerate the taints

1



2

$ kubectl taint nodes NODE_NAME key=value:effect
$ kubectl taint nodes gke-123 service=web:NoExecute

3

```
pima@cloudshell:~/yaml-sample (first-medium-328400)$ cat web.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 3
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web-app
        image: nginx:latest
      tolerations:
      - key: "service-web"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
```

4

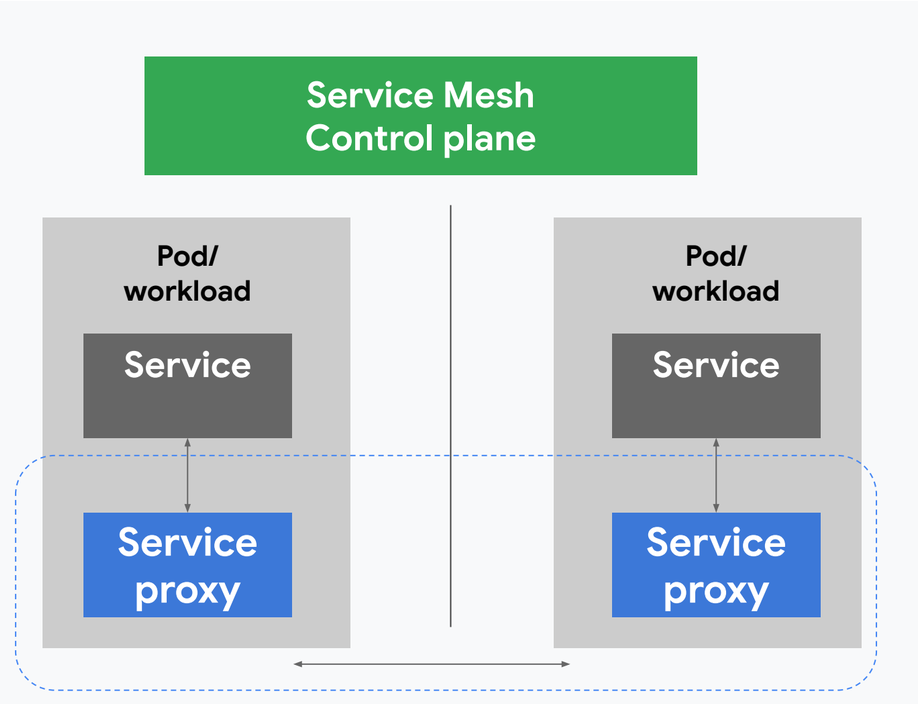
```
FailedScheduling: 0/3 nodes are available: 3 node(s)
had taint {service-web: true}, that the pod didn't
tolerate
```

Service Mesh (Istio / ASM)

Used for visibility, traffic control, security, policy enforcement etc

Outbound features:

- Service authentication
- Load balancing
- Timeouts, retries and circuit breakers
- Connection pool sizing
- Fine-grained routing
- Telemetry
- Request Tracing
- Fault Injection



Data plane

Inbound features:

- Service authentication
- Authorization
- Rate limits
- Load shedding
- Telemetry
- Request Tracing
- Fault Injection

Exam Tip: *Service Mesh (Istio / Anthos Service Mesh) is often the right choice when advanced traffic management is required, eg. mutual TLS, Fault Injection, Traffic Splitting, Circuit Breaking, Connection Pooling etc. Have a look here.*

GOOGLE Kubernetes Engine

#GCPSketchnote

@PVERGADIA THECLOUDGIRL.DEV

9.07.2020

What is GKE?

Your app crashed in production!

SYS ADMIN SAM

Let's fix it once and for all with CONTAINER!

But it worked on my machine?!

DEVELOPER ERIN

CONTAINER FEATURES

- Portable
- Shareable
- Versions
- Isolation
- Introspection
- Fast Deployments
- Immutable
- Reusable

CONTAINER	VIRTUAL MACHINE	BARE-METAL SERVER
Application Code	Application Code	Application Code
Dependencies	Dependencies	Dependencies
Operating System	Operating System	Operating System
Hardware	Hardware	Hardware

But I have got lots of containers, how can I orchestrate all of them?

SAM

I have heard Kubernetes could be the answer...

It is open source platform for managing containers.

ERIN

How does GKE work?

SAM

User

kubectl

http://gRPC/GCP UI

Google Kubernetes Engine Container Cluster in GCP

REGIONAL CONTROL PLANE

- API Server
- Resource Controllers
- Storage
- Scheduler
- Replica in Zone B
- Replica in Zone C

NODES

- User Pod – Containers
- User Pod – Containers
- User Pod – Containers

CONNECTED GCP SERVICES

- VPC Networking
- Persistent Disk
- Load Balancer
- Cloud Monitoring

Color Legend: Google Kubernetes Engine provisions, maintains & operates

Google Kubernetes Engine provisions User optionally maintains & operates

MANAGED KUBERNETES

KUBERNETES → GKE

I bet GKE is the answer with advanced cluster management!

- Easy Cluster Creation
- Load Balancing
- Auto Scaling
- Auto Upgrades
- Auto Repair
- Logging Monitoring

SAM

Kubernetes is not that easy from Installation to Provisioning to Upgrades to Scaling & SLAs

How do I use GKE ?

Cloud Code → Code Repository → Cloud Build → Container Registry → GKE ← Kubectl

WORKLOAD		INFRASTRUCTURE	
Horizontal Pod Autoscaler (HPA)	Vertical Pod Autoscaler (VPA)	Cluster Autoscaler (CA)	Node Auto-Provisioning (NAP)
More Pods	Different Pod Sizes	More Nodes	Different Node Pools

How does it handle traffic growth?

How do I secure apps using GKE ?

SECURE APPS	TRUSTED NETWORKING	SOFTWARE SUPPLY CHAIN SECURITY
Data Encryption	Global VPC	Binary Authorization
Certified Images	Global Load Balancing	Vulnerability Scanning
Private Clusters	Cloud Armor	Managed Base Image
Idenity & Access – IAM – RBAC	Network Policy	