kind: ReplicaSet

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: frontend:v3
```

kind: ReplicaSet

- Define copies
- HPA Declares scaling
- Use Deployment instead

Declarative

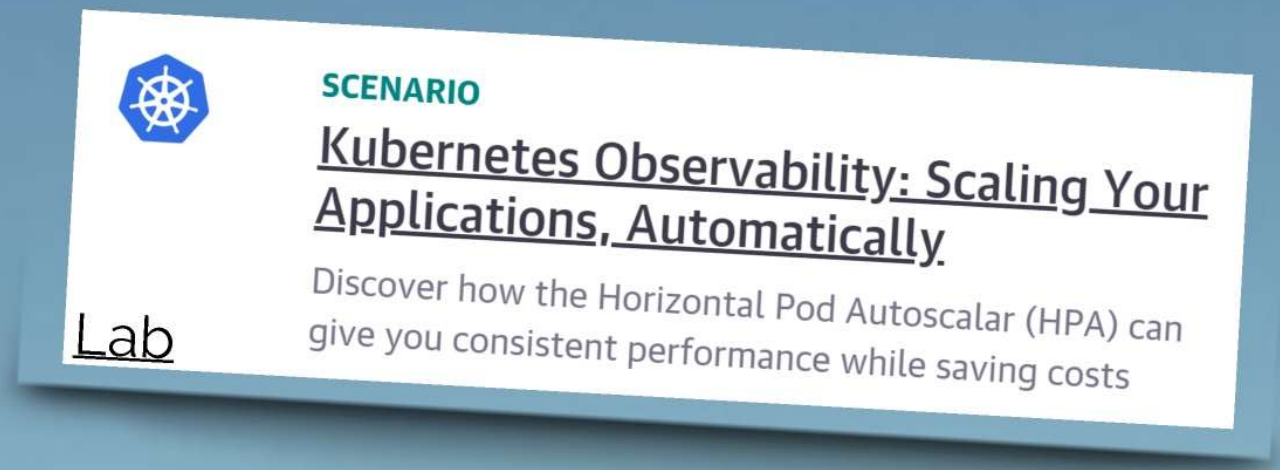
```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Imperative

```
kubectl autoscale rs frontend --
max=10 --min=3 --cpu-percent=50
```

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: frontend:v3
```

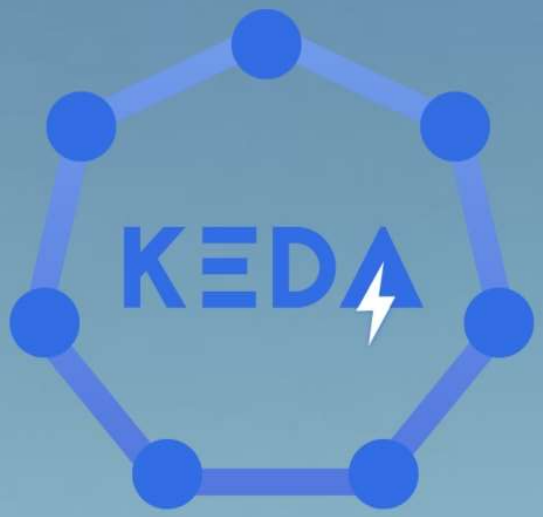
Horizontal Pod Autoscaler



Explore

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

- Install metrics-server for gathering metrics,
- Install pod that can be scaled
- Define scaling rules and number of pods to scale
- Increase service demand to trigger scaling up
- Observe scaling up and down.



Operator for Event-driven Autoscaling

```yaml
apiVersion: keda.k8s.io/v1alpha1
kind: ScaledObject
metadata:
  name: redis-scaledobject
  namespace: my-project
spec:
  scaleTargetRef:
    deploymentName: votes
  triggers:
  - type: redis
    metadata:
      address: REDIS_ADDRESS
      listName: mylist
      listLength: "10"
    authenticationRef:
      name: trigger-auth-redis-secret
```

- Builds on HPA
- Context aware scaling
- Knative integration
- Scale to zero
- 64 scalers: CPU, memory, Metrics, etcd, MySQL, Datadog, Azure monitor, Azure Blob storage, ...

kind: DaemonSet

Pod on every Node

kind: DaemonSet

Pod on every Node

- Logging
- Metrics
- App and resource intelligence

- Taints and tolerations

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      containers:
      - name: fluentd-elasticsearch
        image: fluentd:v2.5.2
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

kind: StatefulSet

- Sticky identity
- Connected volumes
- Distributed managers
- Consistent ID each pod
- Ordered members
- Each has ordinal index
- Ordered startup, reverse shutdown

kind: StatefulSet

- Sticky identity
- Connected volumes
- Distributed managers
- Consistent ID each pod
- Ordered members
- Each has ordinal index
- Ordered startup, reverse shutdown

Examples

Datastores

AMQP
Advanced Message Queuing Protocol

APACHE ACTIVEMQ

APACHE kafka

redis

RabbitMQ

etcd

APACHE ZooKeeper

kind: StatefulSet

- Sticky identity
- Connected volumes
- Distributed managers
- Consistent ID each pod
- Ordered members
- Each has ordinal index
- Ordered startup, reverse shutdown

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

Examples

Datastores

AMQP
Advanced Message Queuing Protocol

APACHE ACTIVEMQ

APACHE kafka

redis

RabbitMQ

etcd

APACHE ZooKeeper

kind: Job

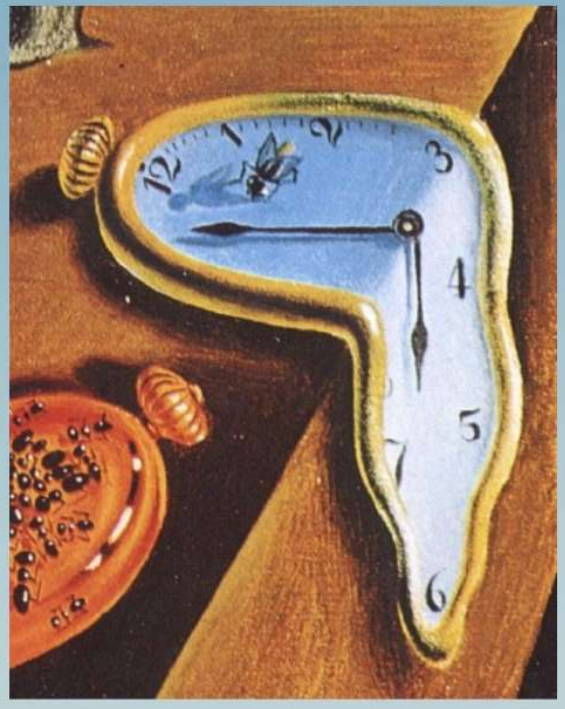
- Triggered from events
- Run in Pods then stop
- Parallel, or not
- Failure policies

- Use patterns
  - Installations
  - Cleanups
  - Builds
  - Datastore updates
  - Helm, Terraform, Liquibase
  - Provisioning
  - Processing

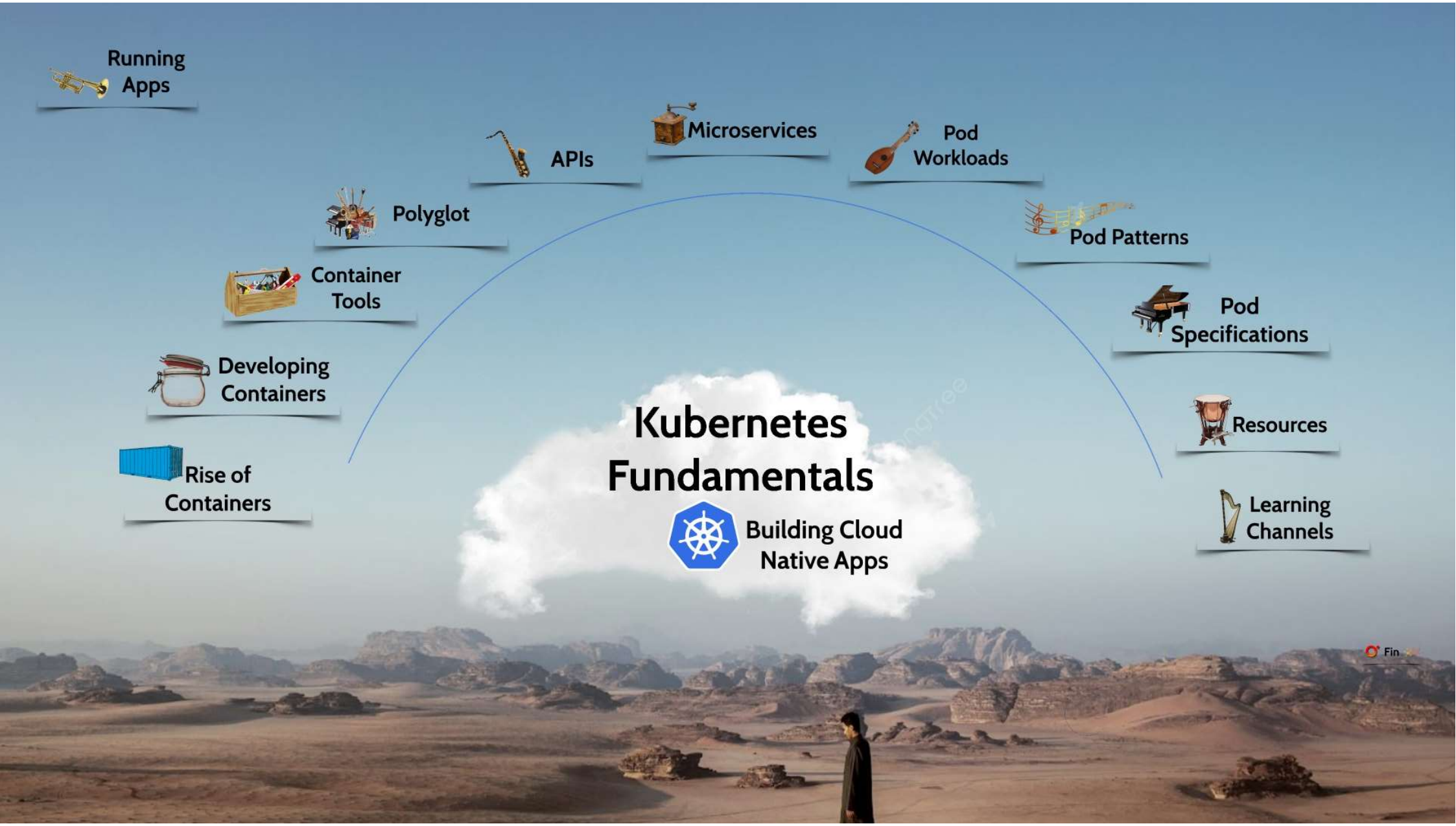
```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  template:
    spec:
      restartPolicy: Never
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-
Mbignum=bpi", "-wle", "print
bpi(2000)"]
```

kind: CronJob

- Reoccurring work
- Periodic
- Trigger based on time



```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello
          restartPolicy: OnFailure
```


Running Apps
APIs
Microservices
Pod Workloads
Polyglot
Pod Patterns
Container Tools
Pod Specifications
Developing Containers
Resources
Rise of Containers
Learning Channels
Kubernetes Fundamentals
Building Cloud Native Apps

Podulation

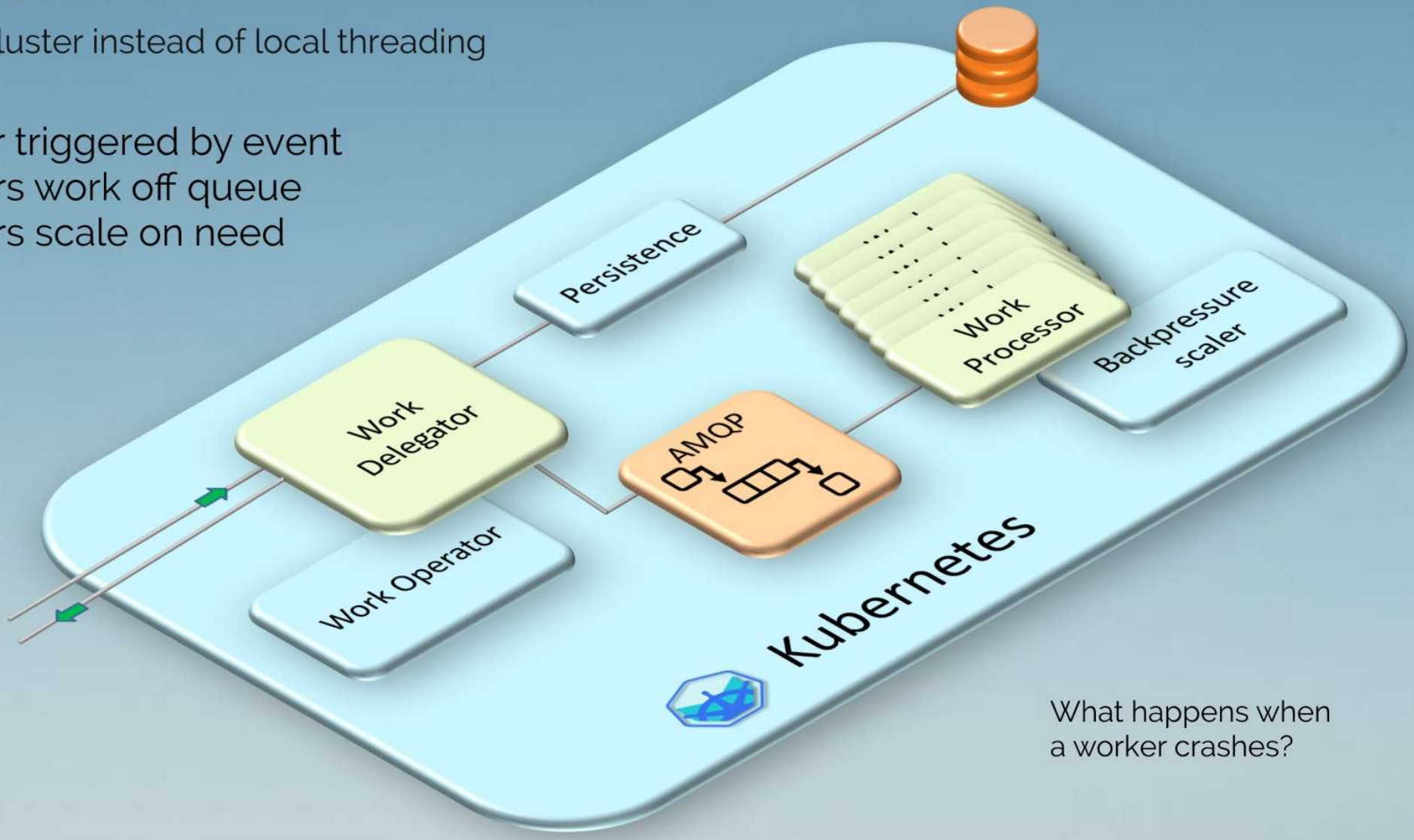
Software Architecture

Patterns and practices
of grouping collections of applications
distributed across a cluster.

Work Queue

Scale with cluster instead of local threading

- Delegator triggered by event
- Processors work off queue
- Processors scale on need



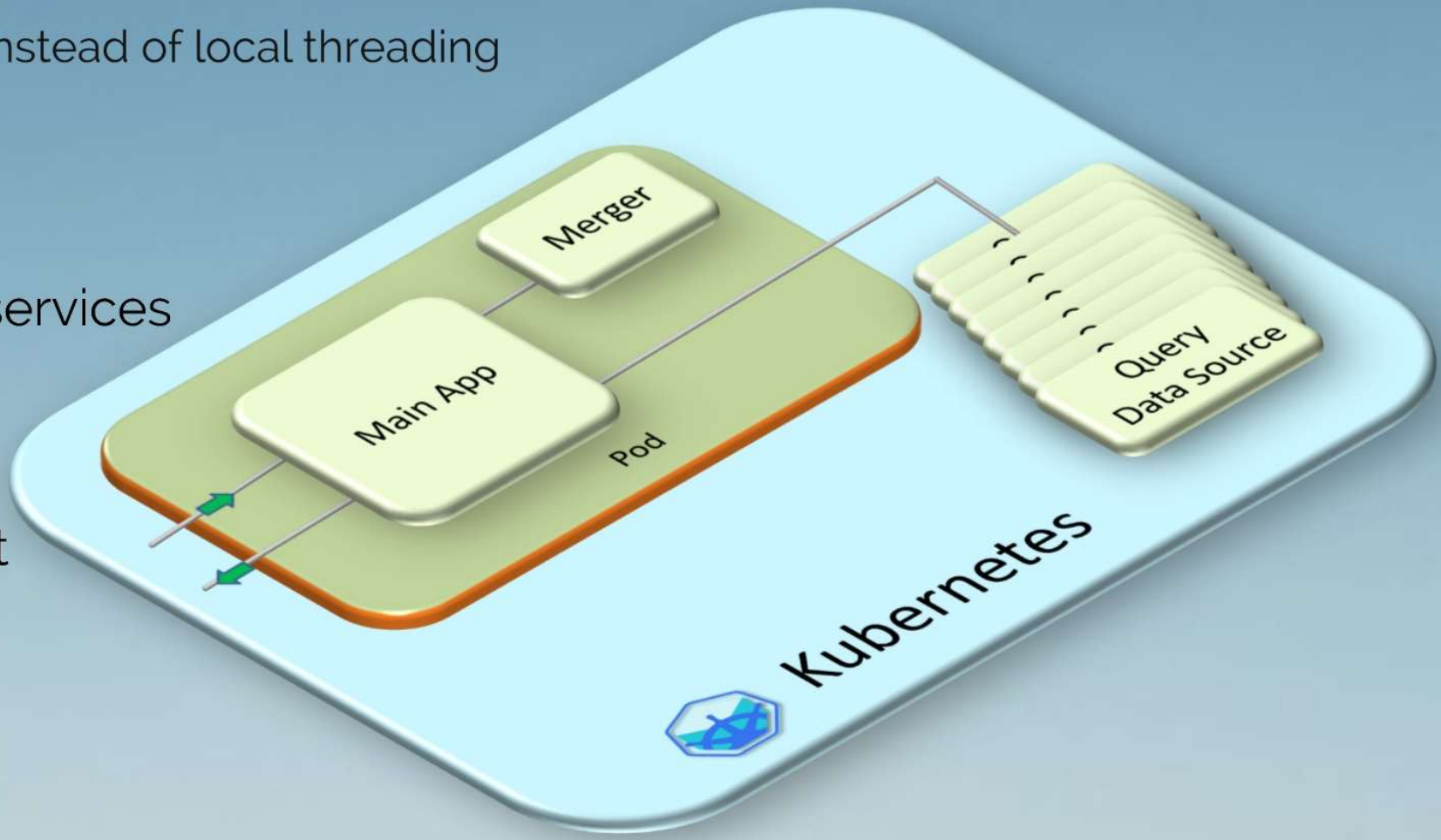
What happens when a worker crashes?

Scatter and Gather

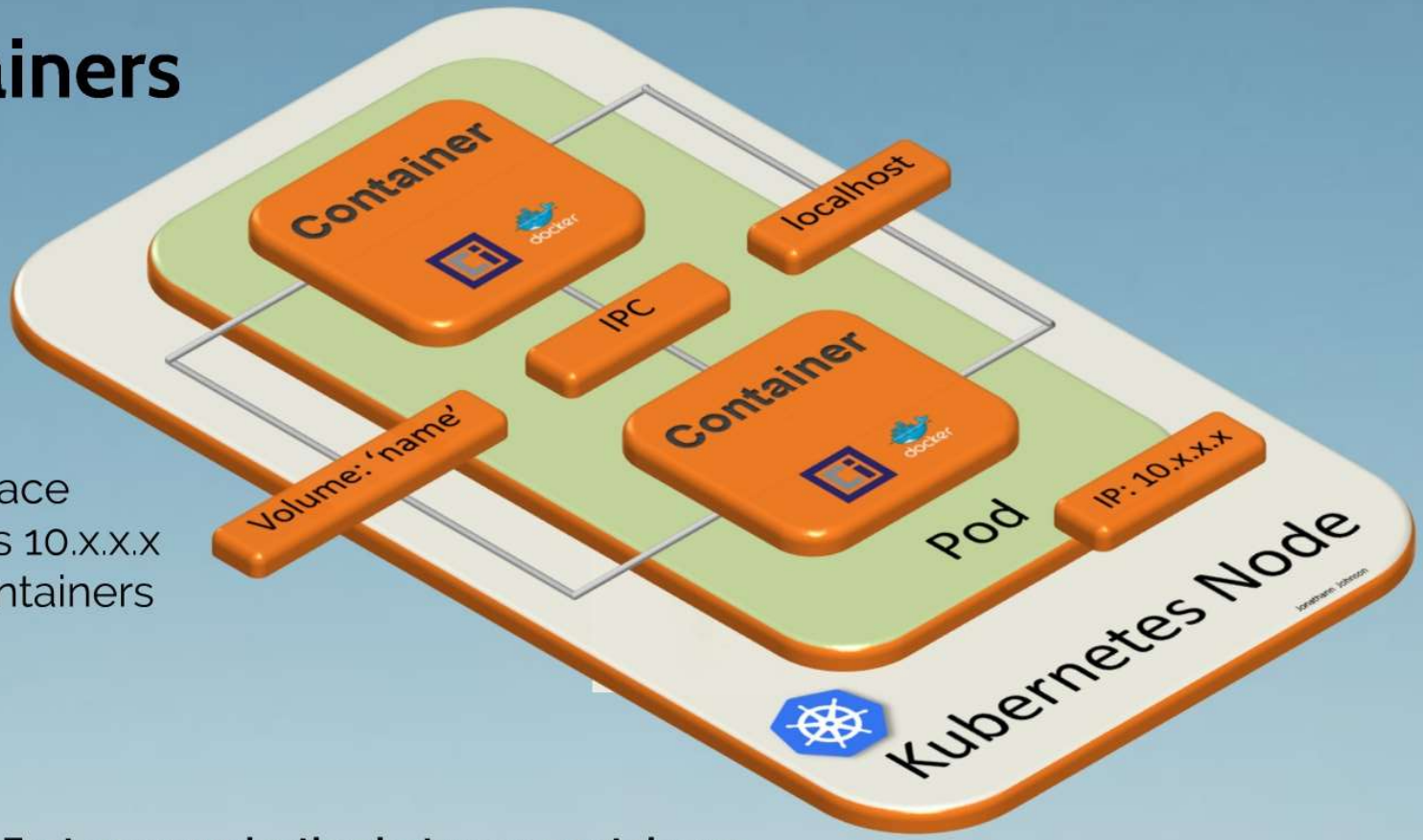
Scale with cluster instead of local threading

- Fan out
- Query multiple services
- Gather data
- Merge results

- Same pod or not



Grouped Containers in Pods

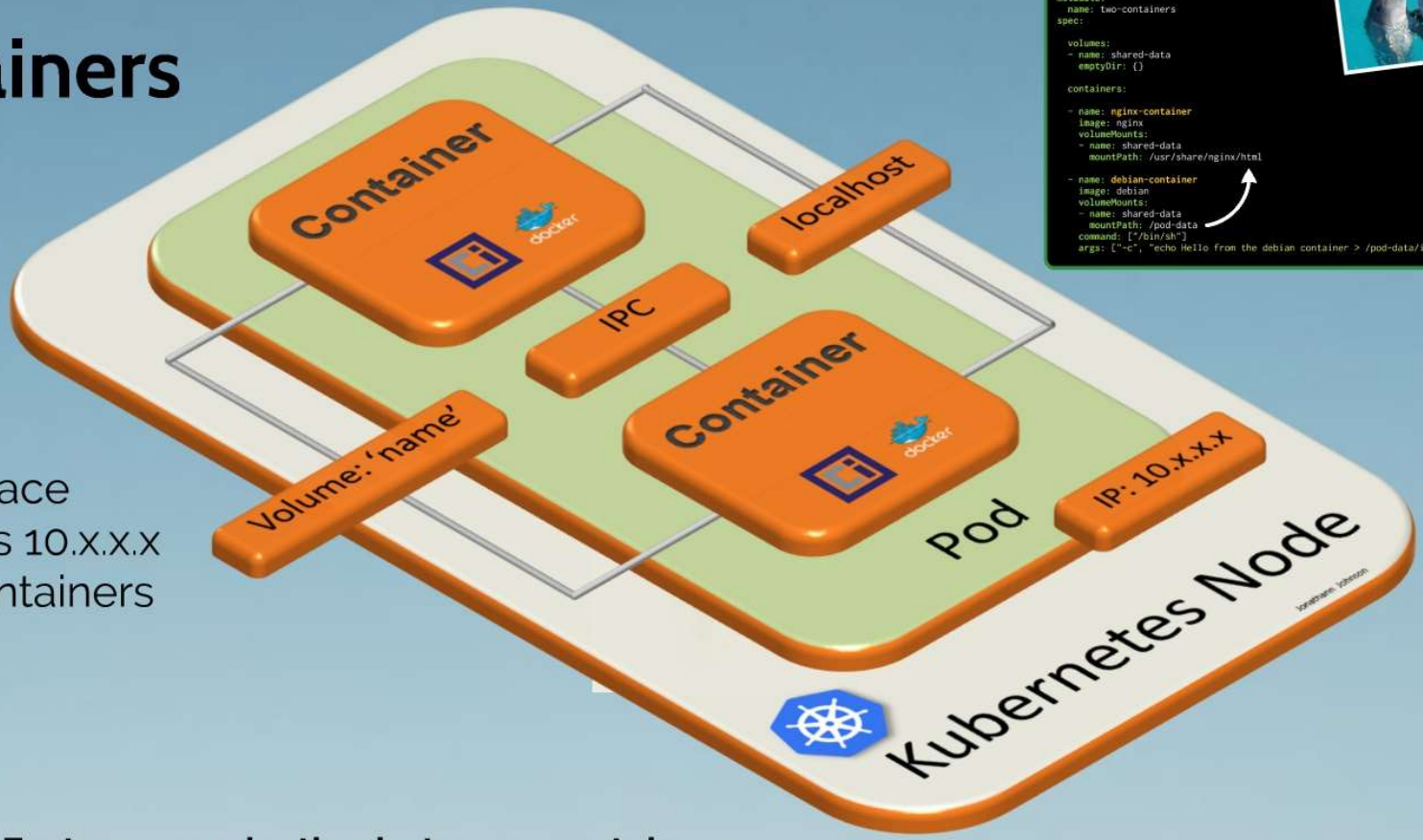


- Pods are atomic units
- Shared Linux namespace
- Assigned IP addresses 10.x.x.x
- localhost between containers
- Shared mounts
- Shared port space

Fast communication between containers

- Shared PersistenceVolume mount: "emptyDir"
- localhost
- Linux inter-process communication (IPC)
- Signals, pipes, sockets, message queues, semaphores, shared memory

Grouped Containers in Pods



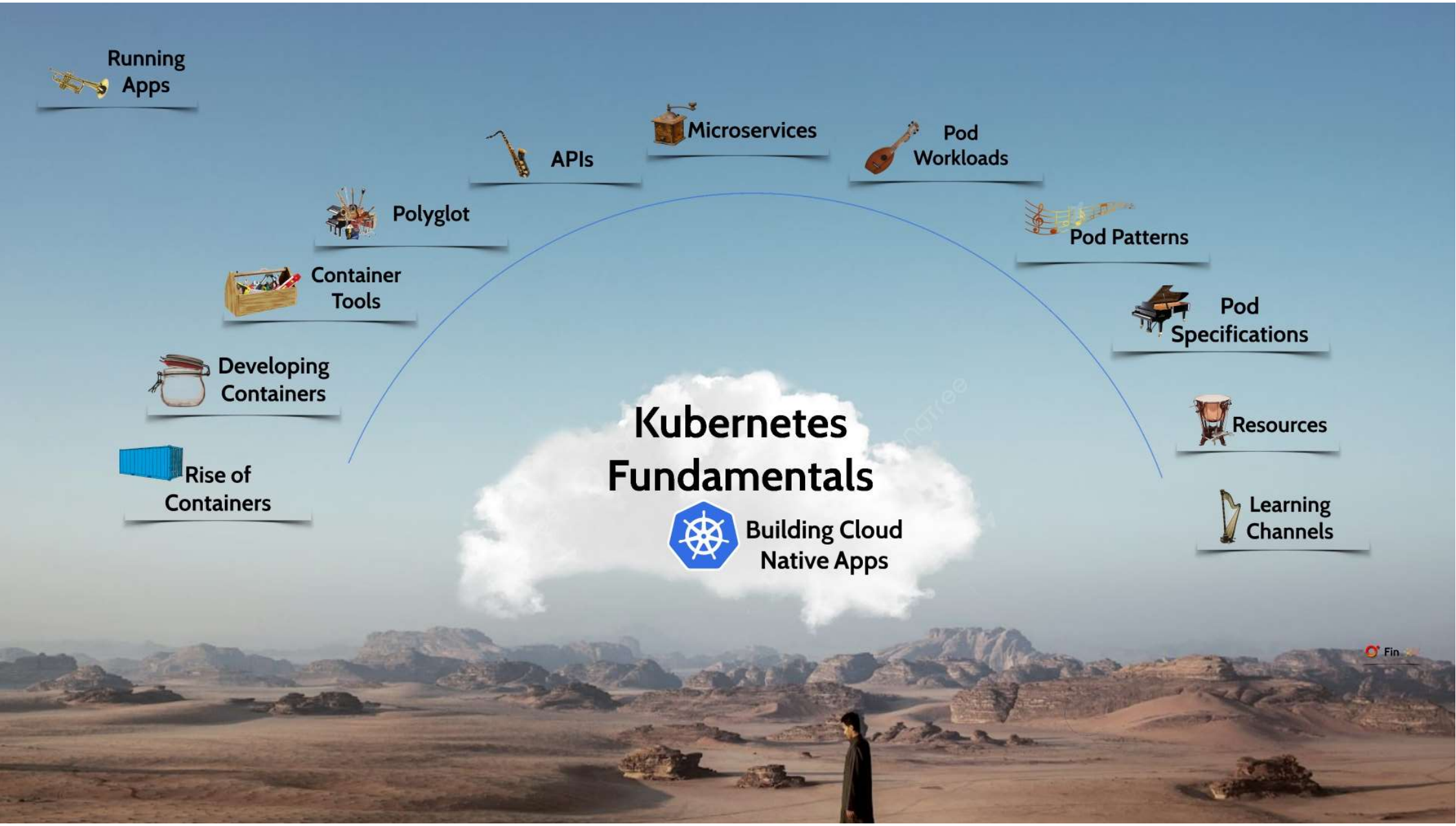

```
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:

  volumes:
  - name: shared-data
    emptyDir: {}

  containers:

  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html

  - name: debian-container
    image: debian
    volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html"]
```

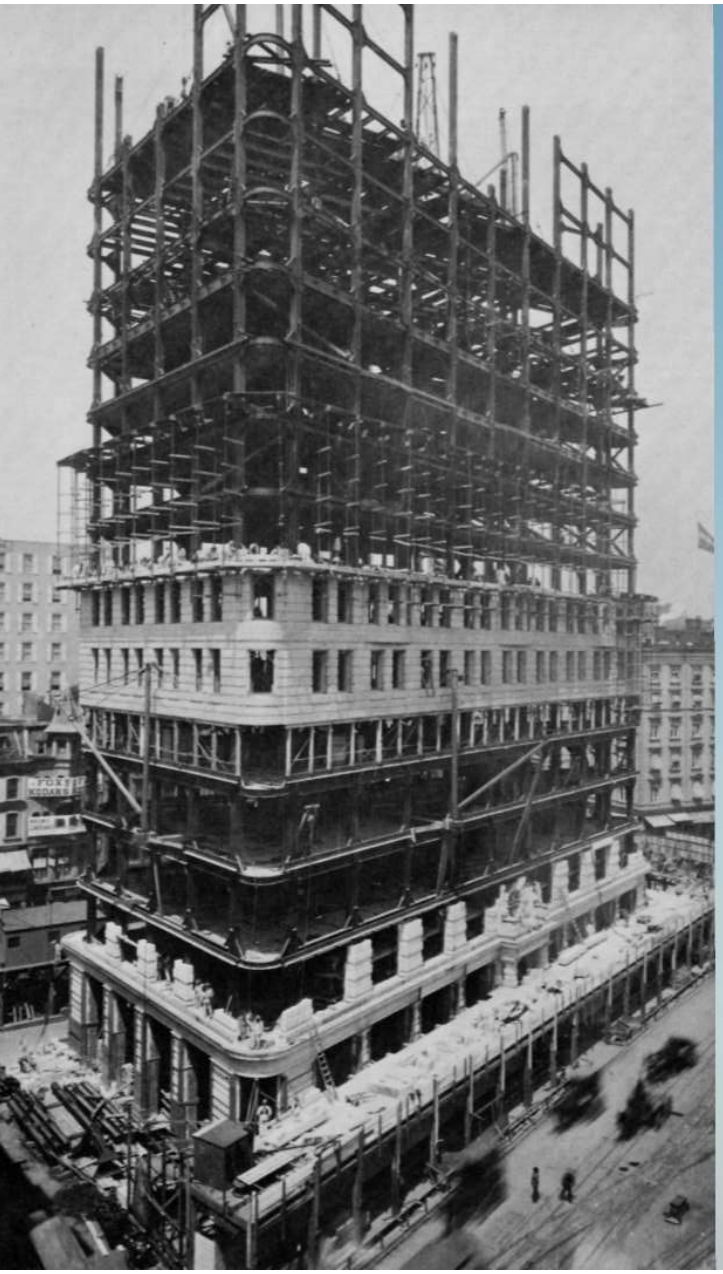
- Pods are atomic units
- Shared Linux namespace
- Assigned IP addresses 10.x.x.x
- localhost between containers
- Shared mounts
- Shared port space

Fast communication between containers

- Shared PersistenceVolume mount: "emptyDir"
- localhost
- Linux inter-process communication (IPC)
- Signals, pipes, sockets, message queues, semaphores, shared memory

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:

  volumes:
  - name: shared-data
    emptyDir: {}

  containers:

  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html

  - name: debian-container
    image: debian
    volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html"]
```


Running Apps
APIs
Microservices
Pod Workloads
Polyglot
Pod Patterns
Container Tools
Pod Specifications
Developing Containers
Resources
Rise of Containers
Learning Channels
Kubernetes Fundamentals
Building Cloud Native Apps



Pod Specifications

- Metadata
- Images
- Context
- Volumes
- Probes
- Quotas
- InitContainers
- Sidecar NEW
- Affinity
- Disruptions
- Ephemeral Containers
- Security

Metadata

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-two-deployment
  annotations:
    devStage: beta
  labels:
    type: webserver
spec:
  selector:
    matchLabels:
      app: nginx-two
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-two
    spec:
      containers:
      - name: nginx-two
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
          name: nginx-pod-port
```

Metadata

Name: unique for resource in namespace

Name: unique for container in pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-two-deployment
  annotations:
    devStage: beta
  labels:
    type: webserver
spec:
  selector:
    matchLabels:
      app: nginx-two
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-two
    spec:
      containers:
      - name: nginx-two
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
```

Metadata

Name: unique for resource in namespace

Annotate: Attach arbitrary non-identifying metadata to objects

Name: unique for container in pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-two-deployment
  annotations:
    devStage: beta
  labels:
    type: webserver
spec:
  selector:
    matchLabels:
      app: nginx-two
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-two
    spec:
      containers:
      - name: nginx-two
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
          name: nginx-pod-port
```

Metadata

Name: unique for resource in namespace

Annotate: Attach arbitrary non-identifying metadata to objects

Labels: For categorizing to match. search, and filter objects

Name: unique for container in pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-two-deployment
  annotations:
    devStage: beta
  labels:
    type: webserver
spec:
  selector:
    matchLabels:
      app: nginx-two
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-two
    spec:
      containers:
      - name: nginx-two
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
```

Metadata

Name: unique for resource in namespace

Annotate: Attach arbitrary non-identifying metadata to objects

Labels: For categorizing to match. search, and filter objects

Namespace: Not often applied in manifest

Name: unique for container in pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-two-deployment
  annotations:
    devStage: beta
  labels:
    type: webserver
spec:
  selector:
    matchLabels:
      app: nginx-two
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-two
    spec:
      containers:
      - name: nginx-two
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
          name: nginx-pod-port
```

Metadata

Name: unique for resource in namespace

Annotate: Attach arbitrary non-identifying metadata to objects

Labels: For categorizing to match. search, and filter objects

Namespace: Not often applied in manifest

Name: unique for container in pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-two-deployment
  annotations:
    devStage: beta
  labels:
    type: webserver
spec:
  selector:
    matchLabels:
      app: nginx-two
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-two
    spec:
      containers:
      - name: nginx-two
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
```

Q: Label vs Annotation?

A: Use annotation when you don't need to search on a label

Community defines

- Recommended Labels
- Well-Known Labels, Annotations and Taints

Images

1 or more containers per pod

Image pulled from repos...

- Reference image tag:version
- imagePullPolicy
- imagePullSecrets (IfNotPresent, Always, Never)

Each Container Has...

- Name
- Ports
- Probes
- Volumes
- Environment vars
- CPU / memory quotas
- Hooks: PostStart & PreStop

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.19.3-alpine
        ports:
        - name: http-port
          containerPort: 80
      imagePullSecrets:
        - name: regcred
```

The Twelve-Factor App

Factor III: Config

Store config in the environment

An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

- Resource handles to the databaseMemcached, and other backing services
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Where to store the environment var for the URL for a microservice connecting to MySQL?

```
MYSQL_URL = jdbc:mysql://host1:33060/sakila
```

Where to store a password to access MySQL?

```
MYSQL_PASSWORD = "shhhhh"
```

Declaring ConfigMaps and Secrets

Data can be linked to:

- Environment variables
- Command line parameters
- File mounts (read-only)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: passable
spec:
  containers:
    - name: question-app
      image: grail-seeker
      env:
        - name: swallow-bird-type
          value: "African"
        - name: QUESTION_ONE
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q1
        - name: QUESTION_TWO
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q2
```

Declaring ConfigMaps and Secrets

Data can be linked to:

- Environment variables
- Command line parameters
- File mounts (read-only)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: questions
data:
  q1: "What is your quest?"
  q2: "What is your name?"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: passable
spec:
  containers:
    - name: question-app
      image: grail-seeker
      env:
        - name: swallow-bird-type
          value: "African"
        - name: QUESTION_ONE
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q1
        - name: QUESTION_TWO
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q2
```

Declaring ConfigMaps and Secrets

Data can be linked to:

- Environment variables
- Command line parameters
- File mounts (read-only)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: questions
data:
  q1: "What is your quest?"
  q2: "What is your name?"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: passable
spec:
  containers:
    - name: question-app
      image: grail-seeker
      env:
        - name: swallow-bird-type
          value: "African"
        - name: QUESTION_ONE
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q1
        - name: QUESTION_TWO
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q2
```

Declaring ConfigMaps and Secrets

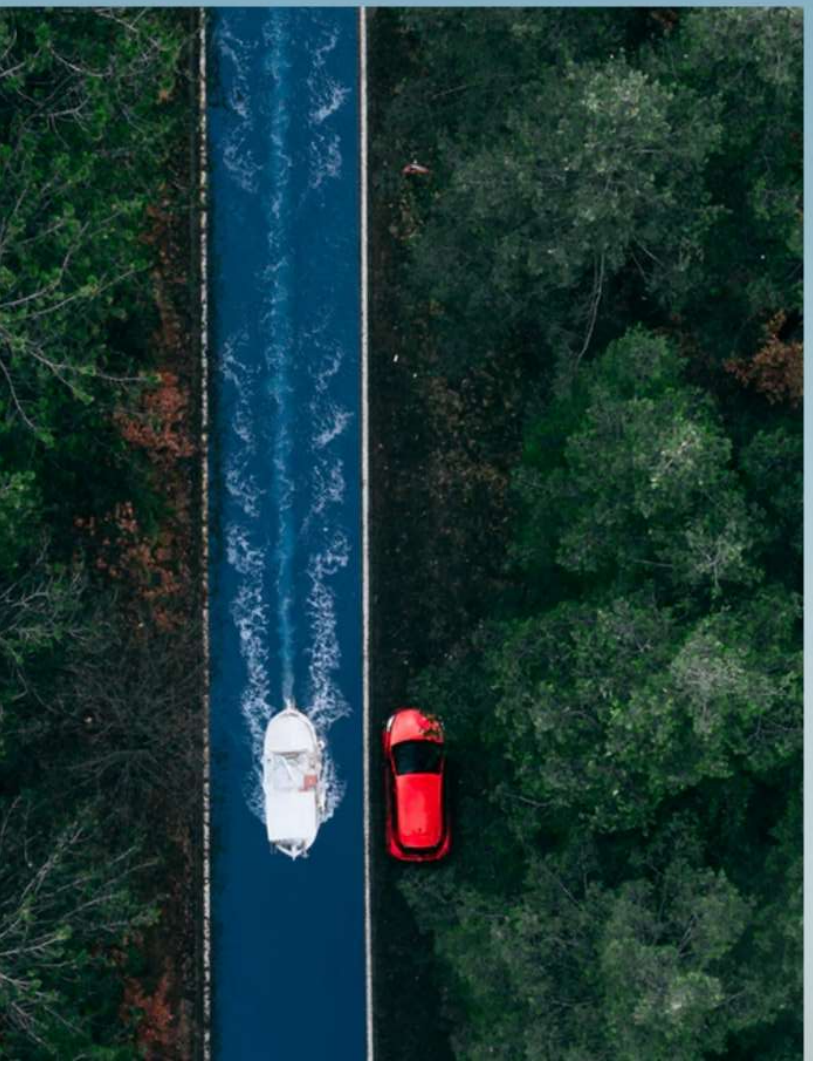
Data can be linked to:

- Environment variables
- Command line parameters
- File mounts (read-only)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: questions
data:
  q1: "What is your quest?"
  q2: "What is your name?"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: passable
spec:
  containers:
    - name: question-app
      image: grail-seeker
      env:
        - name: swallow-bird-type
          value: "African"
        - name: QUESTION_ONE
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q1
        - name: QUESTION_TWO
          valueFrom:
            configMapKeyRef:
              name: questions
              key: q2
```

Context



SCENARIO

Kubernetes Fundamentals: ConfigMaps and Secrets

The fundamentals of ConfigMaps and Secrets

Lab

Explore

- Create configuration data
- Share context across deployments
- 3 techniques Pods access configuration data
- Keeping data outside of code

Access Secrets in Store

CONTAINER
STORAGE
INTERFACE

Secrets Store CSI Driver

The Secrets Store CSI Driver secrets-store.csi.k8s.io allows Kubernetes to mount multiple secrets, keys, and certs stored in enterprise-grade external secrets stores into their pods as a volume.

Several providers

- Akeyless Provider
- AWS Provider
- Azure Provider
- GCP Provider
- Vault Provider

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-provider
spec:
  provider: azure
  parameters:
```

Pod accesses K8s Secret via a file mount.
Auto rotation can set set for Secrets -- should be tested

Secrets with Azure

Set up Azure Keyvault with secrets (az commands)

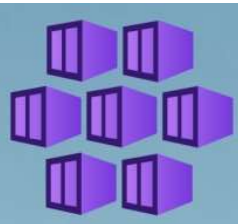
Use the Azure Key Vault provider for Secrets Store CSI Driver in an AKS

Helm: Install Secret Store CIS Driver and Azure Keyvault Provider

- ArtifactHub
- Homepage

Secrets with Azure

Pod accesses Secret via a file mount

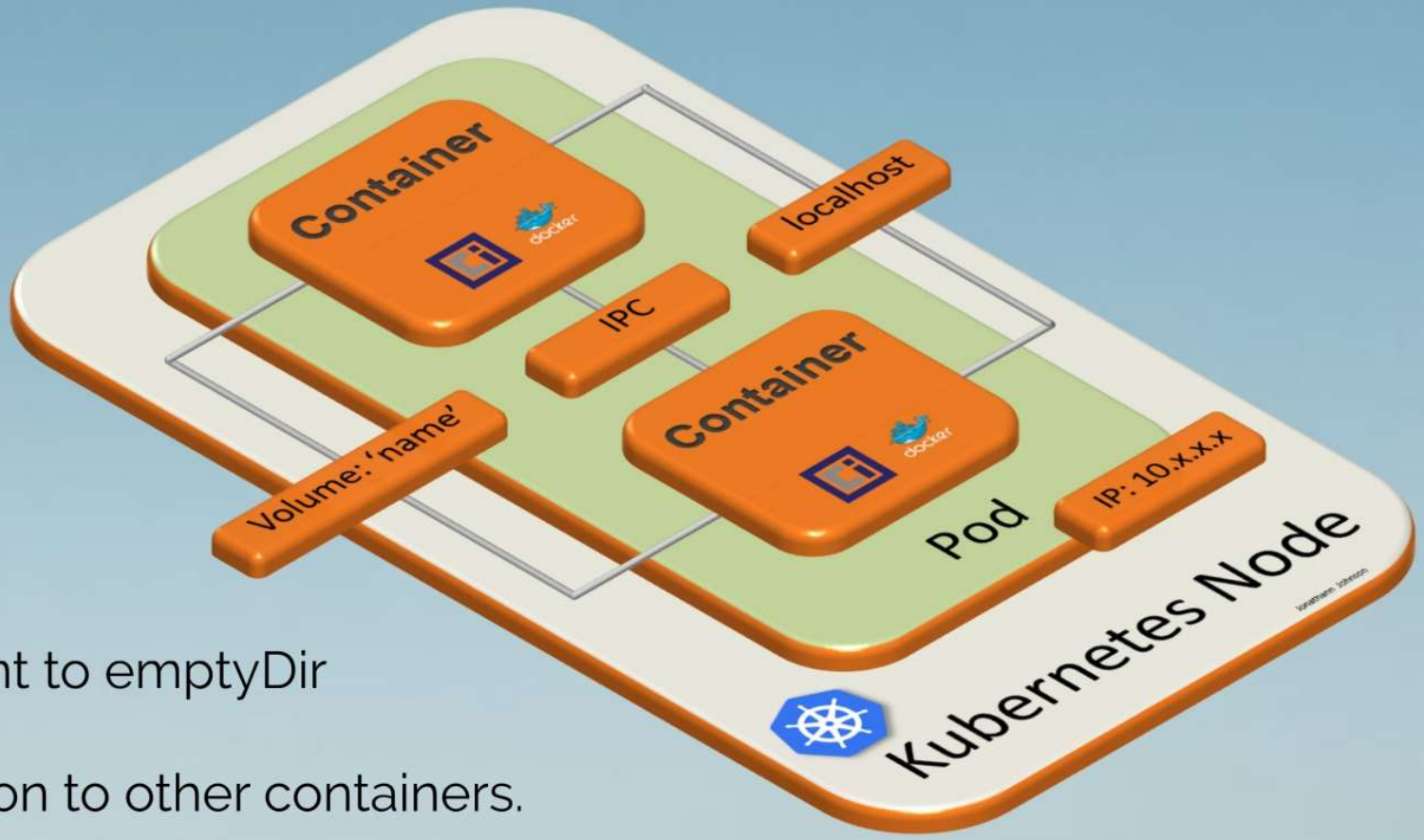


```yaml
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-provider
  namespace: default
spec:
  provider: azure
  parameters:
    usePodIdentity: "false"
    useManagedIdentity: "false"
    keyvaultName: "$KEYVAULT_NAME"
    objects: |
      array:
        - |
          objectName: $SECRET_NAME
          objectType: secret
          objectVersion: $SECRET_VERSION
        - |
          objectName: $KEY_NAME
          objectType: key
          objectVersion: $KEY_VERSION
    tenantId: "$TENANT_ID"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-csi
spec:
  containers:
  - image: alpine
    name: alpine
    command:
      - "sh"
      - "-c"
      - "echo going to sleep... && sleep 10000"
    volumeMounts:
    - name: secrets-store-inline
      mountPath: "/csi-secrets"
      readOnly: true
  volumes:
  - name: secrets-store-inline
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: "my-provider"
```

Volumes

Local Pod volume



Containers can mount to emptyDir
Use as cache
Use as communication to other containers.

Volumes

Local Node volume

Sometime containers need access to Host Volume with **hostPath**

hostPath mounts can typed as:

- DirectoryOrCreate
- **Directory**
- FileOrCreate
- File
- CharDevice
- BlockDevice
- (empty, no check)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: Directory
```

Volumes

Persistent volume

```
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    name: www
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
```

Volumes

Persistent volume

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    name: www
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claim-http
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      storage-context: customers
```

Volumes

Persistent volume

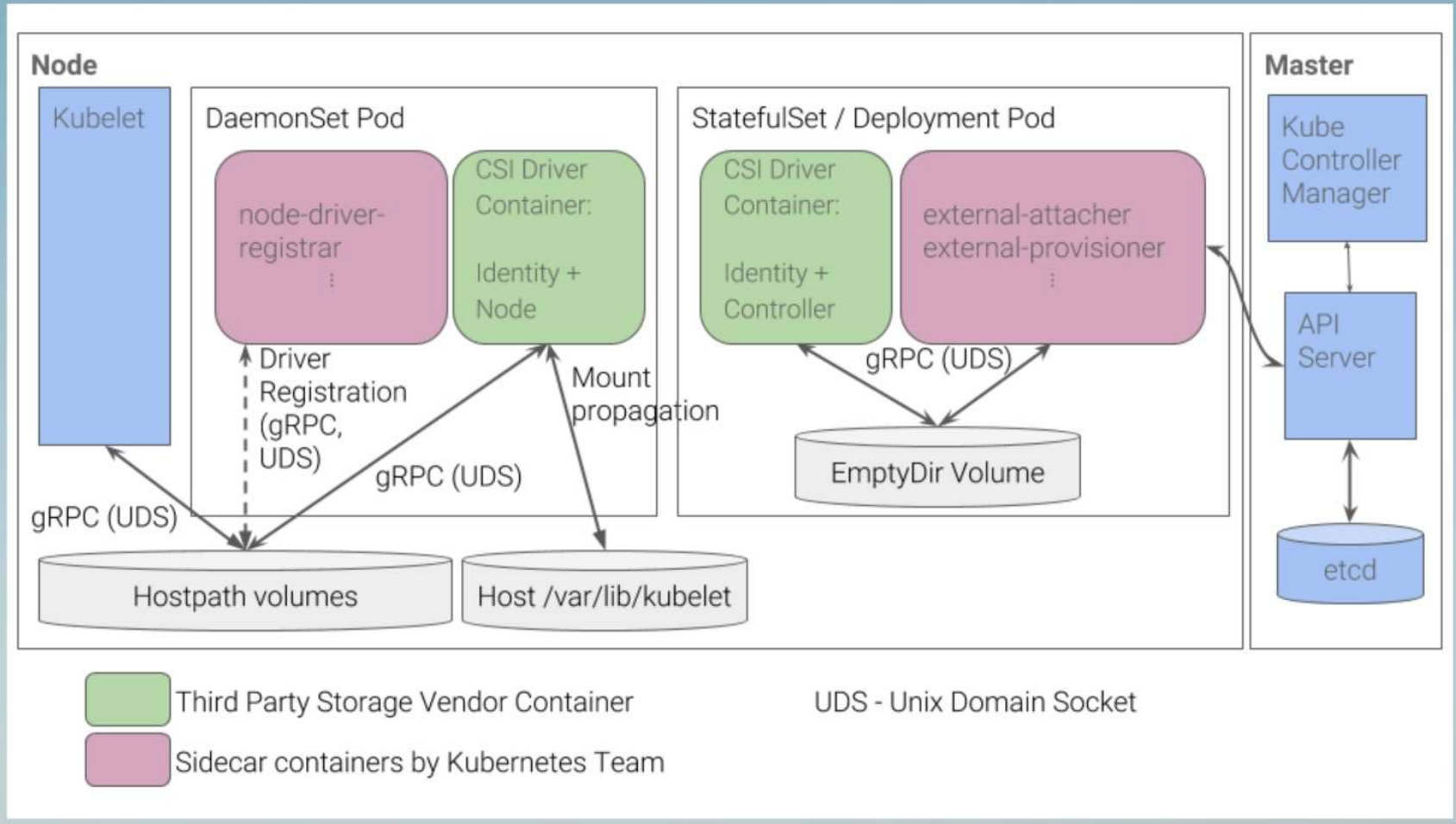
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    name: www
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claim-http
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      storage-context: customers
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-0001
  labels:
    storage-context: customers
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: 172.17.0.48
    path: /exports/data-0001
```

Recommended Mechanism for Deploying CSI Drivers on Kubernetes

CONTAINER STORAGE INTERFACE



https://kubernetes-csi.github.io/docs/deploying.html

https://bit.ly/31T3Ilw

Volumes

Containers, Pods, Nodes,
and Cluster come and go.
Volumes remain

SCENARIO

Kubernetes Fundamentals: Volumes and Mounts

An outward look at the file systems and volumes available to your app.

Lab

Explore

- Setup a volume
- Declare volume access
- Connect Pod to volume

Probes

Checking Container and Pod Health

Lab

- **Liveness**
  Are you even responding?
  If not, Pod is killed (moo)

Probes

Checking Container and Pod Health

Lab

- **Liveness**
  Are you even responding?
  If not, Pod is killed (moo)

- **Readiness**
  Are you available for traffic?
  Service load balancer skips Pod until healthy

Probes

Checking Container and Pod Health

Lab

- **Liveness**
  Are you even responding?
  If not, Pod is killed (moo)

- **Readiness**
  Are you available for traffic?
  Service load balancer skips Pod until healthy

- **Startup (new)**
  Geez, you're taking a long time to initialize!
  Databases, complex models, clunky legacy apps

Probes

Checking Container and Pod Health

SCENARIO

Kubernetes Fundamentals: Health Checks with Probes

Explore the importance and difference between the three probe types for Pods

Lab

- **Liveness**
  Are you even responding?
  If not, Pod is killed (moo)

- **Readiness**
  Are you available for traffic?
  Service load balancer skips Pod until healthy

- **Startup (new)**
  Geez, you're taking a long time to initialize!
  Databases, complex models, clunky legacy apps

Probes

Checking Container and Pod Health

- HTTP Status codes
- TCP Connection
- gRPC, HTTP
- Exec, zero=pass, non zero=fail

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      failureThreshold: 3
      initialDelaySeconds: 45
      periodSeconds: 30
```

Probes

Checking Container and Pod Health

Adjust these behaviors to match expected container personality:

- Pass/fail logic
- Communication type
- initialDelaySeconds
- periodSeconds
- timeoutSeconds
- successThreshold
- failureThreshold

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

A Top Reason Why Clusters Fail

Your applications in containers consume CPU and memory.
Don't keep your resource needs a secret from Kubernetes.

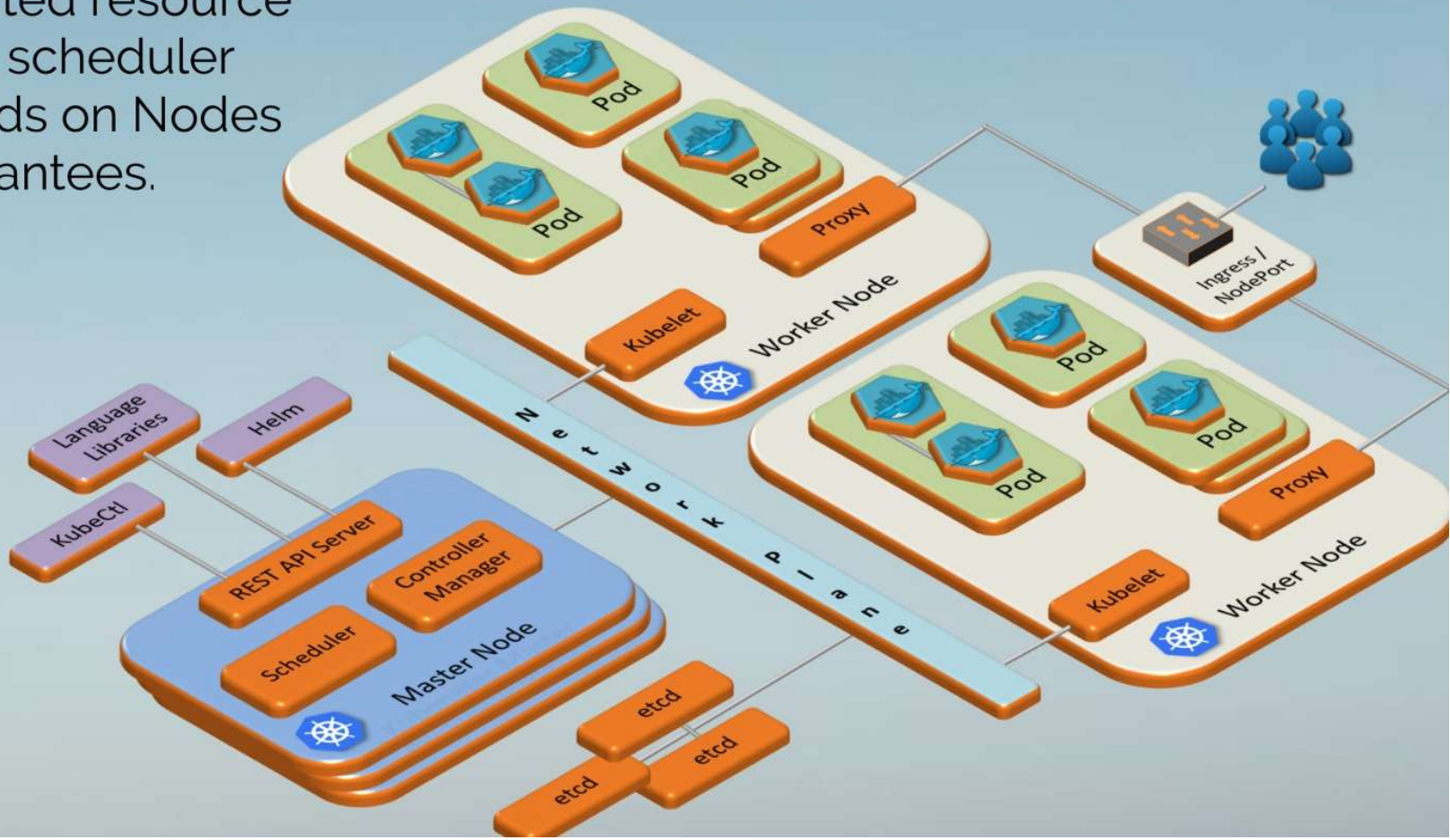
Office workers on floor 7 1/2

Being John Malkovich

1999 American fantasy comedy film directed by Spike Jonze and written by Charlie Kaufman

CPU and Memory are limited resources

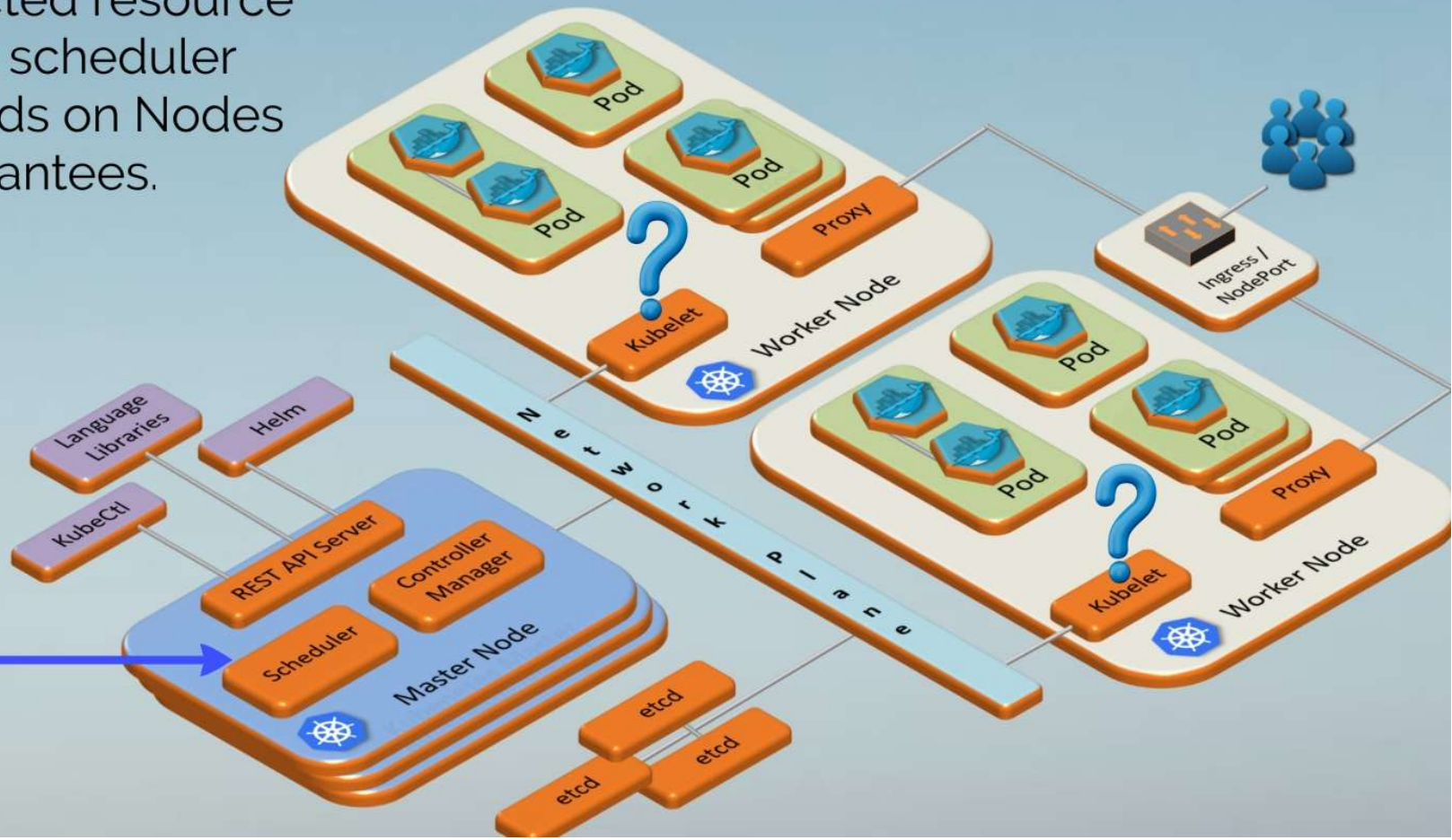
By providing expected resource needs, Kubernetes scheduler "bin packs" your Pods on Nodes with resource guarantees.



CPU and Memory are limited resources

By providing expected resource needs, Kubernetes scheduler "bin packs" your Pods on Nodes with resource guarantees.

Is there room?



Quality of Service
BestEffort

- BestEffort
- Burstable
- Guaranteed

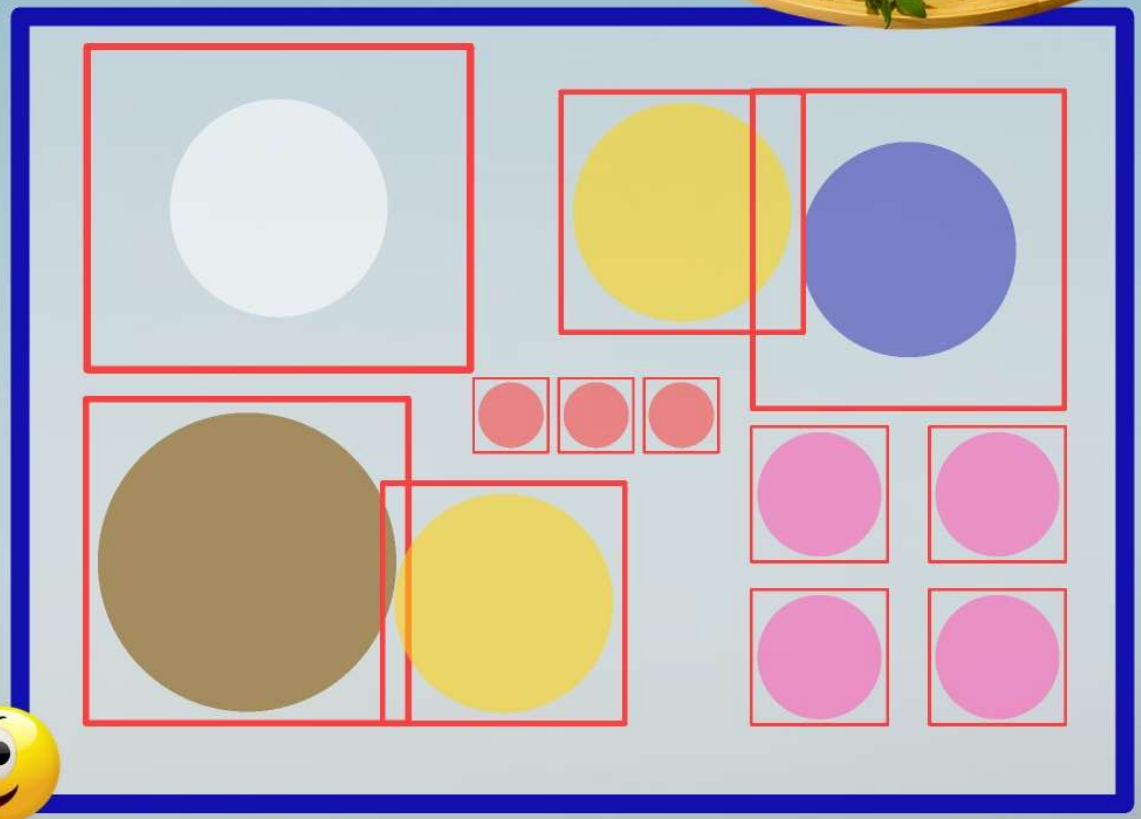
```
apiVersion: v1
kind: Pod
metadata:
  name: pack-em-in
spec:
  containers:
  - name: best-effort-example
    image: potential-beast:1.2.3
```

Dangerous - no CPU or memory requests
Rogue pods disrupt other pods and nodes

Quality of Service
Burstable

- BestEffort
- Burstable
- Guaranteed

```
apiVersion: v1
kind: Pod
metadata:
  name: pack-em-in
spec:
  containers:
  - name: best-effort-example
    image: potential-beast:1.2.3
    resources:
      limits:
        memory: "512Mi"
        cpu: "1024m"
      requests:
        memory: "256Mi"
        cpu: "600m"
```

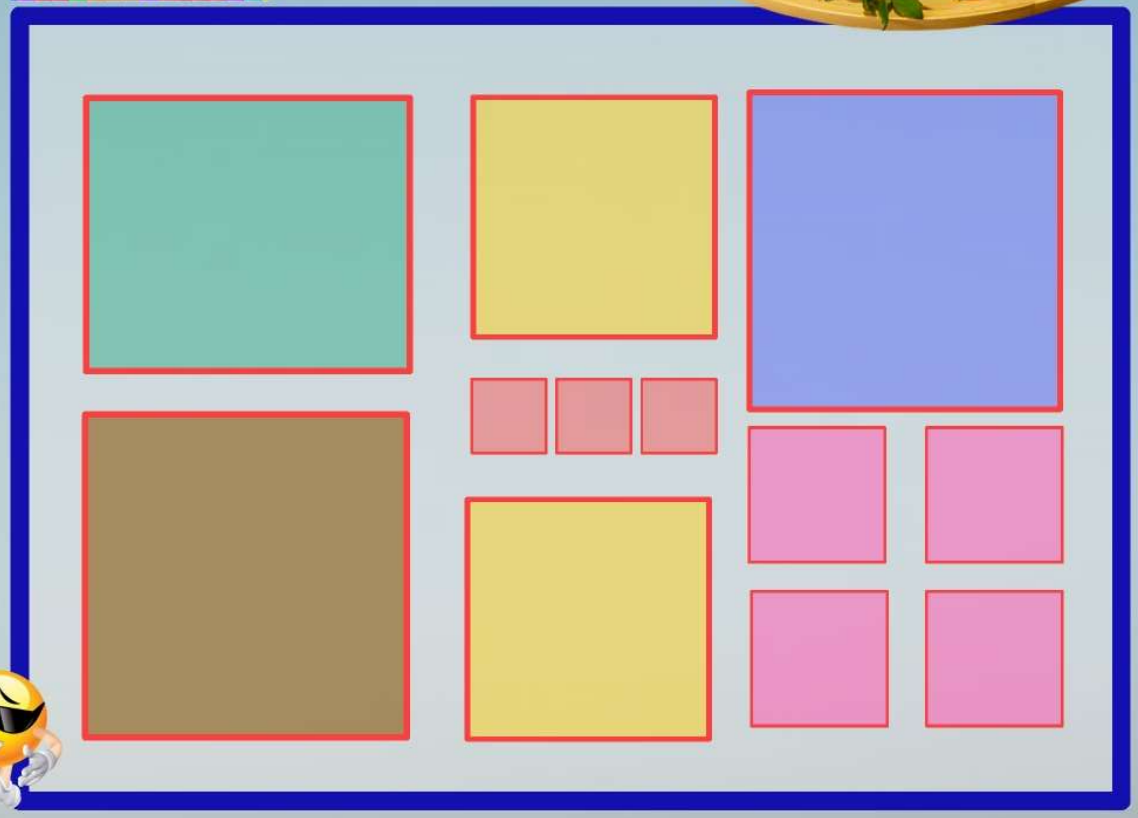


Better, works with autoscalar
Noisy Neighbor problem remains when too many bursts

Quality of Service
Guaranteed

- BestEffort
- Burstable
- Guaranteed

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pack-em-in
spec:
  containers:
  - name: best-effort-example
    image: potential-beast:1.2.3
    resources:
      limits:
        memory: "256Mi"
        cpu: "1024m"
      requests:
        memory: "256Mi"
        cpu: "1024m"
```



Safest

Potentially over allocates and wastes ($)

ResourceQuotas

Apply ResourceQuota to limit available resource for:

- Pods in Namespaces
- Pods in an affinity group

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pods-high
spec:
  hard:
    cpu: "1000"
    memory: 200Gi
    pods: "10"
  scopeSelector:
    matchExpressions:
    - operator : In
      scopeName: PriorityClass
      values: ["high"]
```

Hey, I Like You ... NOT!

3 native match making tricks through labels

Hey, I Like You ... NOT!

3 native match making tricks through labels

Node Affinity/Anti-affinity

Pods are marked to tell scheduler what **Nodes** they prefer, or not.

Hey, I Like You ... NOT!

3 native match making tricks through labels

Node Affinity/Anti-affinity

Pods are marked to tell scheduler what **Nodes** they prefer, or not.

Taints and Tolerations

Nodes are marked to tell scheduler what **Pods** they prefer, or not.

Hey, I Like You ... NOT!

3 native match making tricks through labels

Node Affinity/Anti-affinity

Pods are marked to tell scheduler what **Nodes** they prefer, or not.

Taints and Tolerations

Nodes are marked to tell scheduler what **Pods** they prefer, or not.

Pod Affinity/Anti-affinity

Pods are marked to tell scheduler what other **Pod** types they wish to be near (cloud zones), or not.

Hey, I Like You ... NOT!

3 native match making tricks through labels

Node Affinity/Anti-affinity

Pods are marked to tell scheduler what **Nodes** they prefer, or not.

Taints and Tolerations

Nodes are marked to tell scheduler what **Pods** they prefer, or not.

Pod Affinity/Anti-affinity

Pods are marked to tell scheduler what other **Pod** types they wish to be near (cloud zones), or not.

GPU Node + Python Pod

```
kubectl label nodes <node-name>
accelerator=nvidia-tasla-p100
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
    - name: cuda-vector-add
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1
  nodeSelector:
    accelerator: nvidia-tesla-p100
```