Sharing Directory/File

- We can specific directory or file or directory from host to container as follows:

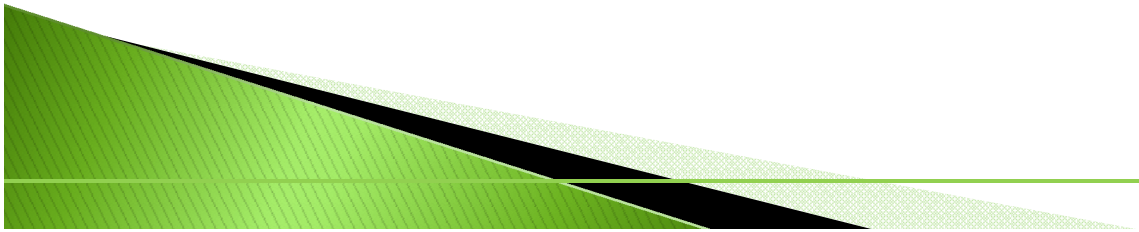
  docker run -d -v /path/on/the/host:/path/in/container image

- Example:

  docker run -itd -v /var/lib:/usr/local -p 9090:8080 --name helloWebApp tomcat

Session: 11

Kubernetes



Atgen

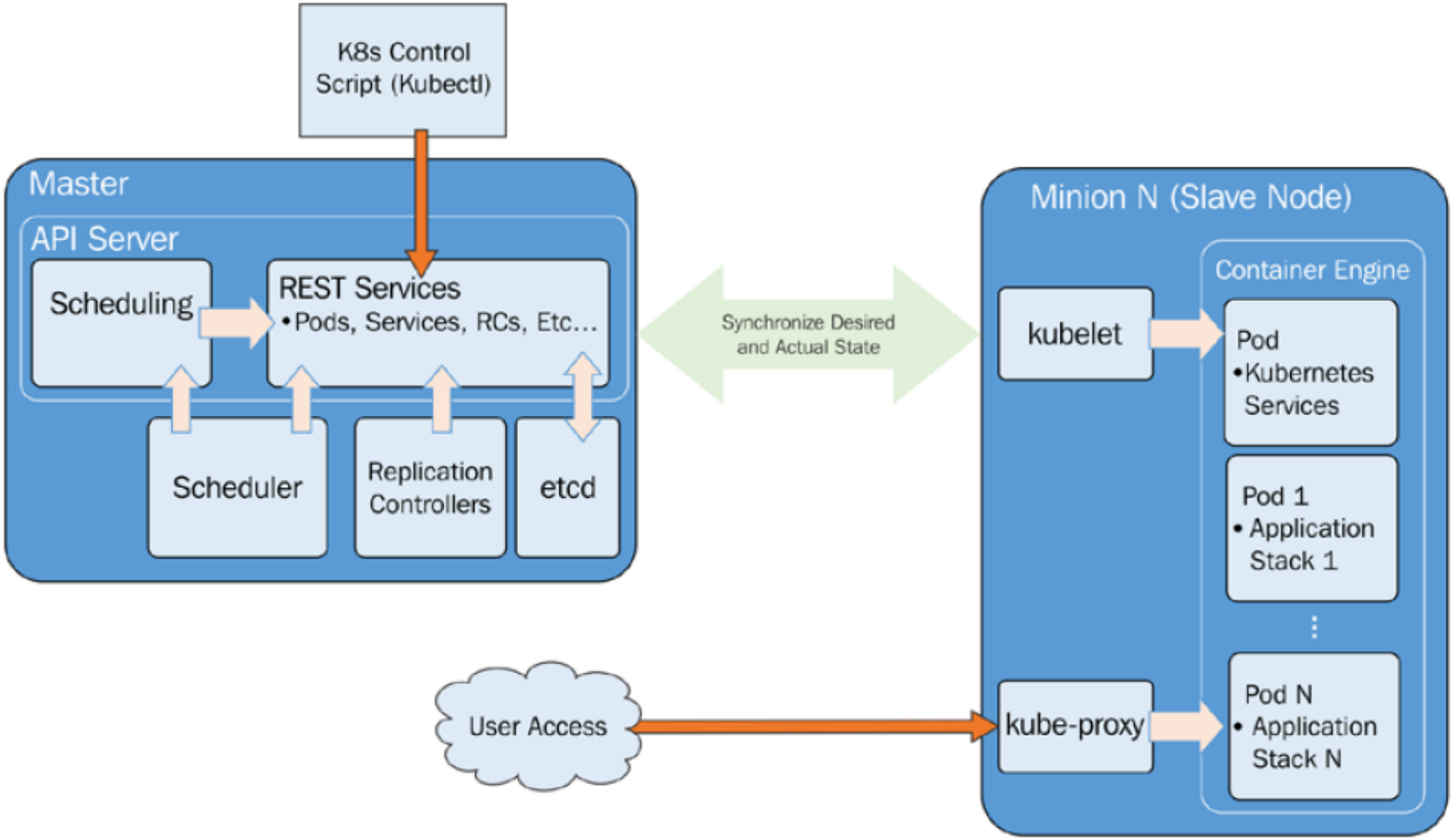
What is Kubernetes

- Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.
- Google open-sourced the Kubernetes project in 2014.
- Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale, combined with best-of-breed ideas and practices from the community.

What is Kubernetes

- Kubernetes brings assistance to orchestrating containers at scale as well as managing full application stacks.
- K8s moves up the stack giving us constructs to deal with management at the application or service level.
- This gives us automation and tooling to ensure high availability, application stack, and service-wide portability.
- K8s also allows finer control of resource usage, such as CPU, memory, and disk space across our infrastructure.
- Kubernetes provides this higher level of orchestration management by giving us key constructs to combine multiple containers, endpoints, and data into full application stacks and services.
- K8s then provides the tooling to manage the when, where, and how many of the stack and its components.

Kubernetes – Core Architecture



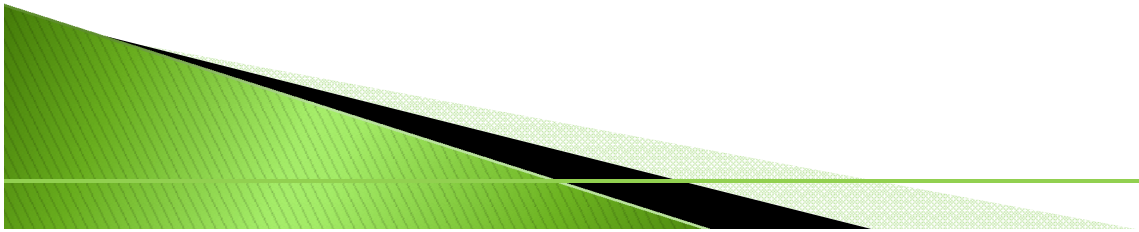
Atgen

Kubernetes – Master

- MASTER is the brain of our cluster
- Following are key components in master:
  - API Server
  - Scheduler
  - Replication Controller
  - etcd

Master – API Server

- API server maintains RESTful web services for querying and defining our desired cluster and workload state.
- It's important to note that the control pane only accesses the master to initiate changes and not the nodes directly.



Master – Scheduler

- Scheduler works with the API server to schedule workloads in the form of pods on the actual minion nodes.
- These pods include the various containers that make up our application stacks.
- By default, the basic Kubernetes scheduler spreads pods across the clusters and uses different nodes for matching pod replicas.
- Kubernetes also allows specifying necessary resources for each container, so scheduling can be altered by these additional factors.

Master – Replication Controler

- The replication controller works with the API server to ensure that the correct number of pod replicas are running at any given time.
- This is exemplary of the desired state concept. If our replication controller is defining three replicas and our actual state is two copies of the pod running, then the scheduler will be invoked to add a third pod somewhere on our cluster.
- The same is true if there are too many pods running in the cluster at any given time.
- In this way, K8s is always pushing towards that desired state.

Master – etcd

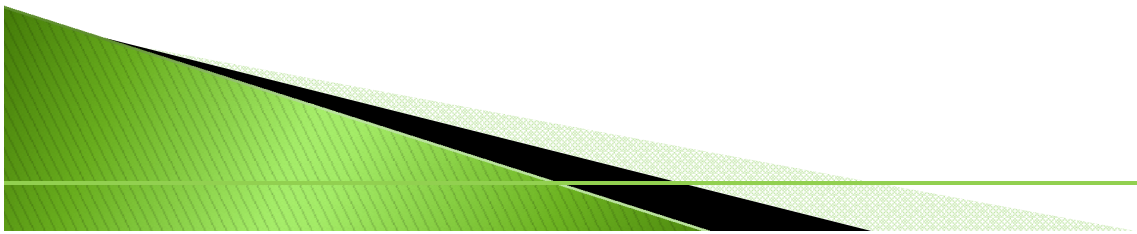
- ‘etcd’ running as a distributed configuration store.
- The Kubernetes state is stored here and ‘etcd’ allows values to be watched for changes.
- Think of this as the brain's shared memory.

Node (formerly minions)

- In each node, we have a couple of components.
- The ‘kubelet’ interacts with the API server to update state and to start new workloads that have been invoked by the scheduler.
- ‘kube-proxy’ provides basic load balancing and directs traffic destined for specific services to the proper pod.
- We also have some default pods, which run various infrastructure services for the node. These pods include services for Domain Name System (DNS), logging, and pod health checks. The default pod will run alongside our scheduled pods on every node.

Session: 12

Deep Dive – Kubernetes



Atgen

Kubernetes – A 'pod'

- Pods allow you to keep related containers close in terms of the network and hardware infrastructure.
- Data can live near the application, so processing can be done without incurring a high latency from network traversal.
- Similarly, common data can be stored on volumes that are shared between a number of containers.
- Pods essentially allow you to logically group containers and pieces of our application stacks together.
- While pods may run one or more containers inside, the pod itself may be one of many that is running on a Kubernetes (minion) node.
- So, pods give us a logical group of containers that we can then replicate, schedule, and balance service endpoints across.

‘pod’ – Example

- Create a file ‘nodejs-pod.yaml’ with below contents:

```
apiVersion: v1
kind: Pod
metadata:
  name: node-js-pod
spec:
  containers:
  - name: node-js-pod
    image: bitnami/apache:latest
    ports:
      - containerPort: 80
```

‘pod’ – Example

- Create a pod using following command:

```
kubectl create -f nodejs-pod.yaml
```

- See details about pod:

```
kubectl describe pods/node-js-pod
```

- Run a command inside pod:

```
kubectl exec node-js-pod -- curl <private ip address>
```

Kubernetes – Deployment

- A Deployment controller provides declarative updates for Pods and ReplicaSets.
- Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Deployment – Example

- Create a file ‘nginx-deployment.yaml’:

```
 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:1.7.9
        ports:
        - containerPort: 80
```

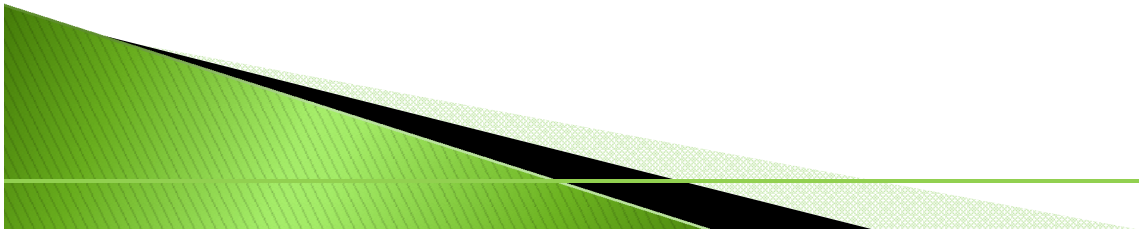
Deployment – Example

- Create a deployment using command:

```
kubectl create -f nginx-deployment.yaml
```

- See details about pod:

```
kubectl get deployments
```

Atgen

Deployment – Example

- Now, update nginx image to '1.9.1' version:

```
kubectl edit deployment/nginx-deployment
```

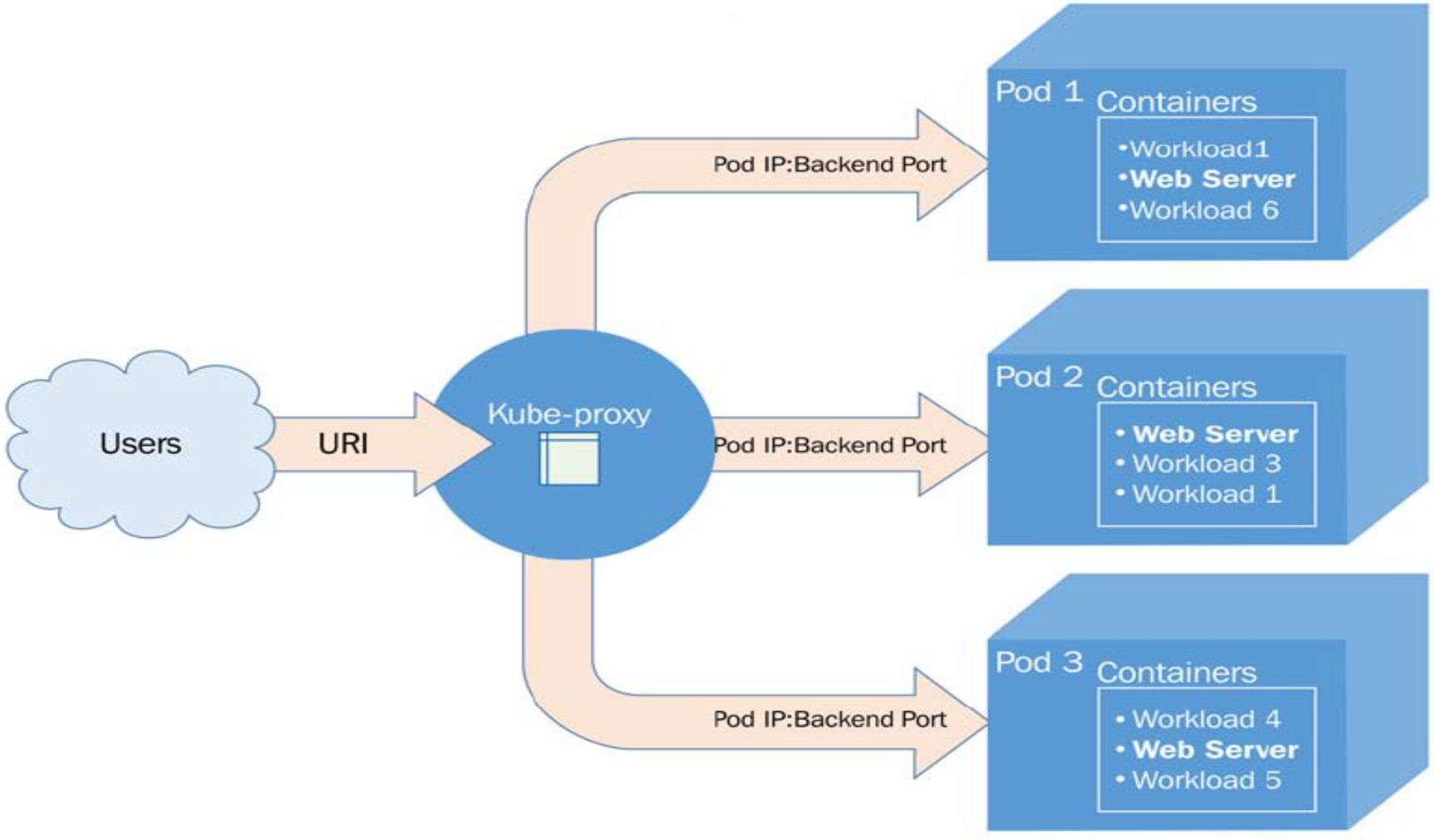
- To see rollout status:

```
kubectl rollout status deployment/nginx-deployment
```

Kubernetes – Services

- Services allow us to abstract access away from the consumers of our applications.
- Using a reliable endpoint, users and other programs can access pods running on your cluster seamlessly.
- K8s achieves this by making sure that every node in the cluster runs a proxy named kube-proxy. As the name suggests, kube-proxy's job is to proxy communication from a service endpoint back to the corresponding pod that is running the actual application.
- A virtual IP address and port are used as the entry point for the service, and traffic is then forwarded to a random pod on a target port defined.
- Service definitions are monitored and coordinated from the K8s cluster master and propagated to the kube-proxy daemons running on each node.

Kubernetes – Services



Atgen

Kubernetes Application – Example

- Create a file 'nodejs-controller.yaml' with below contents:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: node-js
  labels:
    name: node-js
    deployment: demo
spec:
  replicas: 3
  selector:
    name: node-js
  deployment: demo
  template:
    metadata:
      labels:
        name: node-js
        deployment: demo
    spec:
      containers:
      - name: node-js
        image: jonbaier/node-express-info:latest
        ports:
        - containerPort: 80
```

Kubernetes Application – Example

- Create a file ‘nodejs-rc-service.yaml ‘ with below contents:

```
apiVersion: v1
kind: Service
metadata:
  name: node-js
  labels:
    name: node-js
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    name: node-js
```

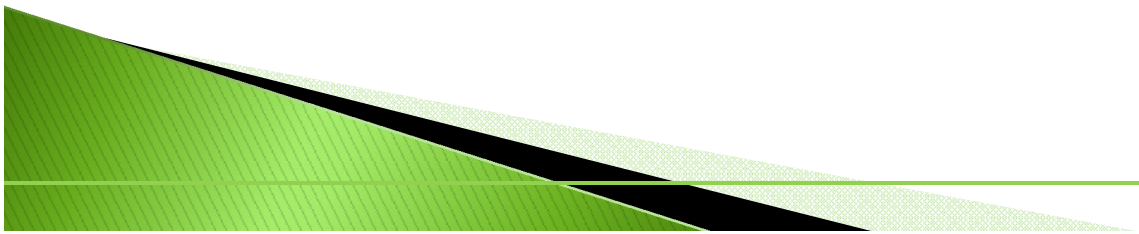
Kubernetes Application – Example

- Create the Node.js express replication controller:

```
kubectl create -f nodejs-controller.yaml
```

- Create the Node.js service:

```
kubectl create -f nodejs-rc-service.yaml
```

Atgen

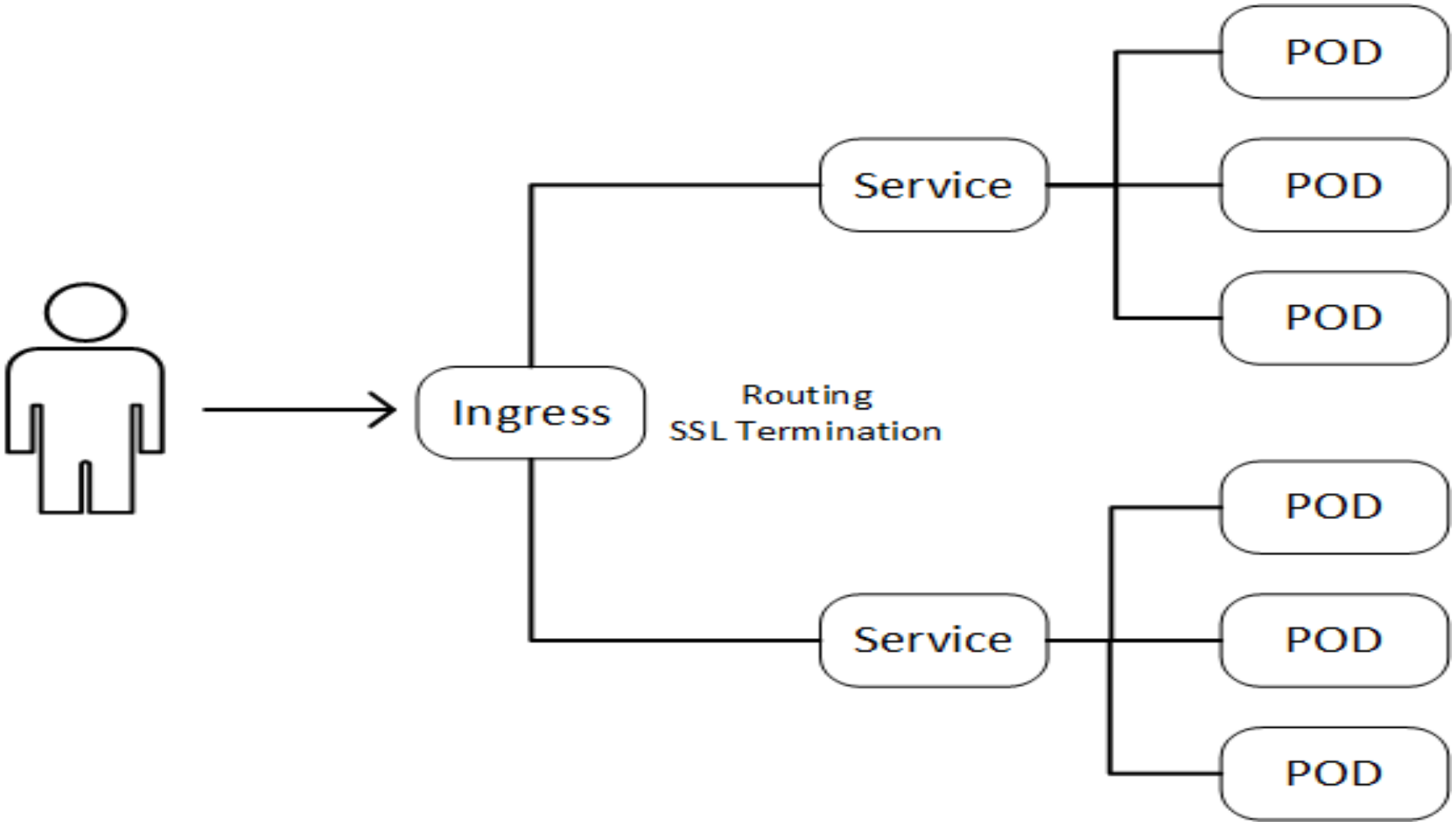
Kubernetes Application – Example

- Let's look at running services:

```
kubectl get services
```

- Now, see if we can connect by opening up the public address in a browser.

Kubernetes – Ingress



Atgen

Kubernetes – Ingress

- Ingress sits in front of multiple services and act as a “smart router” or entrypoint into your cluster.
- The ingress controller will spin up a HTTP(S) Load Balancer for you. This will let you do both path based and subdomain based routing to backend services.
- There are many types of Third Party Ingress controllers, from the Google Cloud Load Balancer, Nginx, Contour, Istio, and more. There are also plugins for Ingress controllers, like the cert-manager, that can automatically provision SSL certificates for your services.
- Ingress is the most useful if you want to expose multiple services under the same IP address, and these services all use the same L7 protocol (typically HTTP).
- You only pay for one load balancer if you are using the native GCP integration, and because Ingress is “smart” you can get a lot of features out of the box (like SSL, Auth, Routing, etc)

Kubernetes Ingress – Example

- Deploy a Web Application:

  kubectl run web --image=gcr.io/google-samples/hello-app:1.0 --port=8080

- Expose Deployment as Service:

  kubectl expose deployment web --target-port=8080 --type=NodePort

- Create an Ingress Resource:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
spec:
  backend:
    serviceName: web
    servicePort: 8080
```

- Deploy a Ingress Resource:

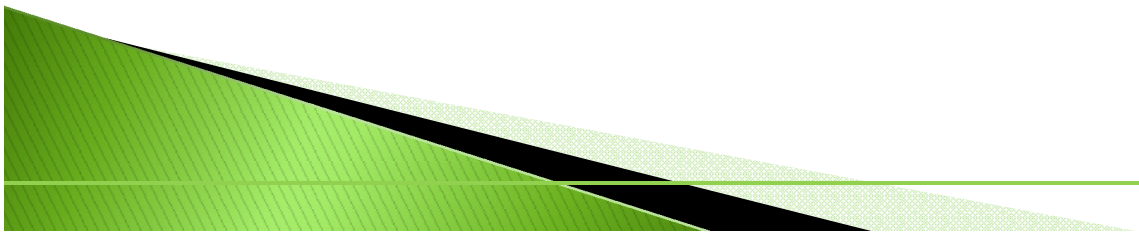
  kubectl apply -f basic-ingress.yaml

- Access Ingress on External IP

Atgen

Session: 13

Managing Kubernetes



Atgen

Kubernetes – Namespace

- Kubernetes *namespaces* help different projects, teams, or customers to share a Kubernetes cluster.
- By default, a Kubernetes cluster will instantiate a default namespace when provisioning the cluster to hold the default set of Pods, Services, and Deployments used by the cluster.

```
kubectl get namespaces
```

Kubernetes – Namespace

- Let's create a "production" name space.
- Create a file "production.json" with below:

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
      "name": "production",
      "labels": {
        "name": "production"
       }
   }
}
```

- Execute:

  kubectl create –f production.json

Kubernetes – Namespace

- Create deployment in namespace:

  kubectl run snowflake --image=kubernetes/serve_hostname --replicas=2 --namespaces=production

Kubernetes – RBAC

- RBAC uses the **rbac.authorization.k8s.io** API group to drive authorization decisions, allowing admins to dynamically configure policies through the Kubernetes API.
- In the RBAC API, a role contains rules that represent a set of permissions.
- A role can be defined within a namespace with a **Role**

Kubernetes – RBAC

- Create a Namespace

  kubectl create namespace testproject

- Make Directory for SSL keys/certificates

  mkdir rbac && cd rbac

- Generate openssl key

  openssl genrsa -out testuser.key 2048

- Generate CSR

  openssl req -new -key testuser.key -out testuser.csr
  -subj "/CN=testuser/O=testproject"

Kubernetes – RBAC

- Check Kubernetes Certificates Location

  ls /etc/kubernetes/pki

- Sign certificates

  openssl x509 -req -in testuser.csr
  -CA /etc/kubernetes/pki/ca.crt
  -CAkey /etc/kubernetes/pki/ca.key
  -CAcreateserial -out testuser.crt -days 500

Kubernetes – RBAC – Role

- A Role can only be used to grant access to resources within a single namespace. Here's an example :

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: testproject
  name: testuser-role
rules:
- apiGroups: ["", "extensions", "apps"] # "" indicates the core API group
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete“]
```

Kubernetes – RBAC – RoleBinding

- A role binding grants the permissions defined in a role to a user or set of users. Permissions can be granted within a namespace with a RoleBinding. Here's an example :

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: testuser-rolebinding
  namespace: testproject
subjects:
  - kind: User # May be "User", "Group" or "ServiceAccount"
    name: testuser
    apiGroup: ""
roleRef:
  kind: Role
  name: testuser-role
  apiGroup: ""
```

Kubernetes – RBAC

- Set User Credentials

  kubectl config set-credentials testuser
  --client-certificate=$HOME/rbac/testuser.crt
  --client-key=$HOME/rbac/testuser.key

- Set Context for User

  kubectl config set-context testuser-context
  --cluster=kubernetes --namespace=testproject
  --user=testuser

- Generate config file manually and share testuser.crt/testuser.key/config with client to access K8s.

Kubernetes – Persistent Volume

- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator.
- It is a resource in the cluster just like a node is a cluster resource.
- PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.
- This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

Kube – Persistent Volume Claim

- A PersistentVolumeClaim (PVC) is a request for storage by a user.
- It is similar to a pod. Pods consume node resources and PVCs consume PV resources.
- Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

Kubernetes – Persistent Volume

- Create an index.html file on your Node

  mkdir /mnt/data
  echo 'Hello from Kubernetes storage' > /mnt/data/index.html

Kubernetes – Persistent Volume

- Create a Persistent Volume

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Kube - Persistent Volume Claim

- Create a Persistent Volume

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Kube – Persistent Volume Claim

- Create a Pod that used PVC

```
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          targetPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Kubernetes – HealthChecks

- Let's create a new Controller with HealthChecks:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: node-js
  labels:
    name: node-js
spec:
  replicas: 3
  selector:
    name: node-js
  template:
    metadata:
      labels:
        name: node-js
    spec:
      containers:
      - name: node-js
        image: jonbaier/node-express-info:latest
        ports:
          - containerPort: 80
        livenessProbe:
          httpGet:
            path: /status/
            port: 80
          initialDelaySeconds: 30
          timeoutSeconds: 1
```


Atgen

Kubernetes – HealthChecks

- LivenessProbe options:

```
livenessProbe:
  exec:
    command:
    -/usr/bin/checkServce.sh
  initialDelaySeconds: 90
  timeoutSeconds: 1
```

```
livenessProbe:
  tcpSocket:
    port: 80
  initialDelaySeconds: 15
  timeoutSeconds: 1
```

Kubernetes – Enabling Dashboard

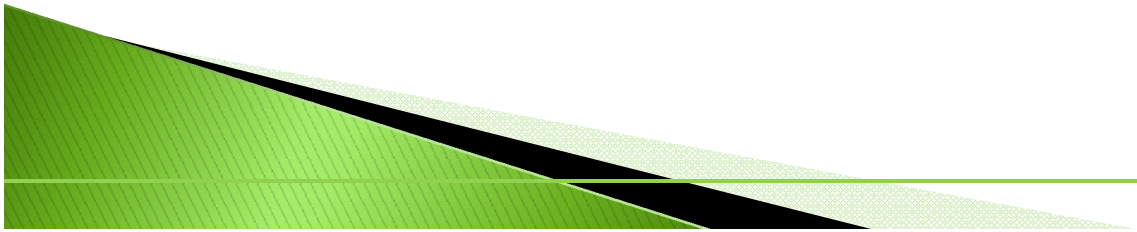
- Create an ADMIN User:

```
kubectl create serviceaccount cluster-admin-dashboard-sa

kubectl create clusterrolebinding cluster-admin-dashboard-sa --clusterrole=cluster-admin
--serviceaccount=default:cluster-admin-dashboard-sa

kubectl get secret | grep cluster-admin-dashboard-sa

kubectl describe secret cluster-admin-dashboard-sa-token-6xm8l
```

Atgen

Kubernetes – Enabling Dashboard

- Opening the Dashboard:

kubectl proxy

http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

Kubernetes – API's

- Opening the Dashboard:

kubectl proxy --port=8080 &

curl http://localhost:8080/api/

curl http://localhost:8080/api/v1/namespaces/{namespace}/pods

Kubernetes – Priority Class

- A PriorityClass is a non-namespaced object that defines a mapping from a priority class name to the integer value of the priority.
- Example Priority Class:

```
apiVersion: scheduling.k8s.io/v1alpha1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

Kubernetes – Priority Class

- Pod using Priority Class:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Contact us

- Contact @ http://www.atgensoft.com/
- Linkedin: @atgensoft-solutions-llp
- Twitter: @skedautomation
- FaceBook: @atgensoftsolutions
- Our Products & Services:
  - DevOps Training, Consulting & Implementation
  - Sked Automation Software
  - Ajar DBMS
  - Automation Design & Solutions

Atgen

Thank You !!

Atgen