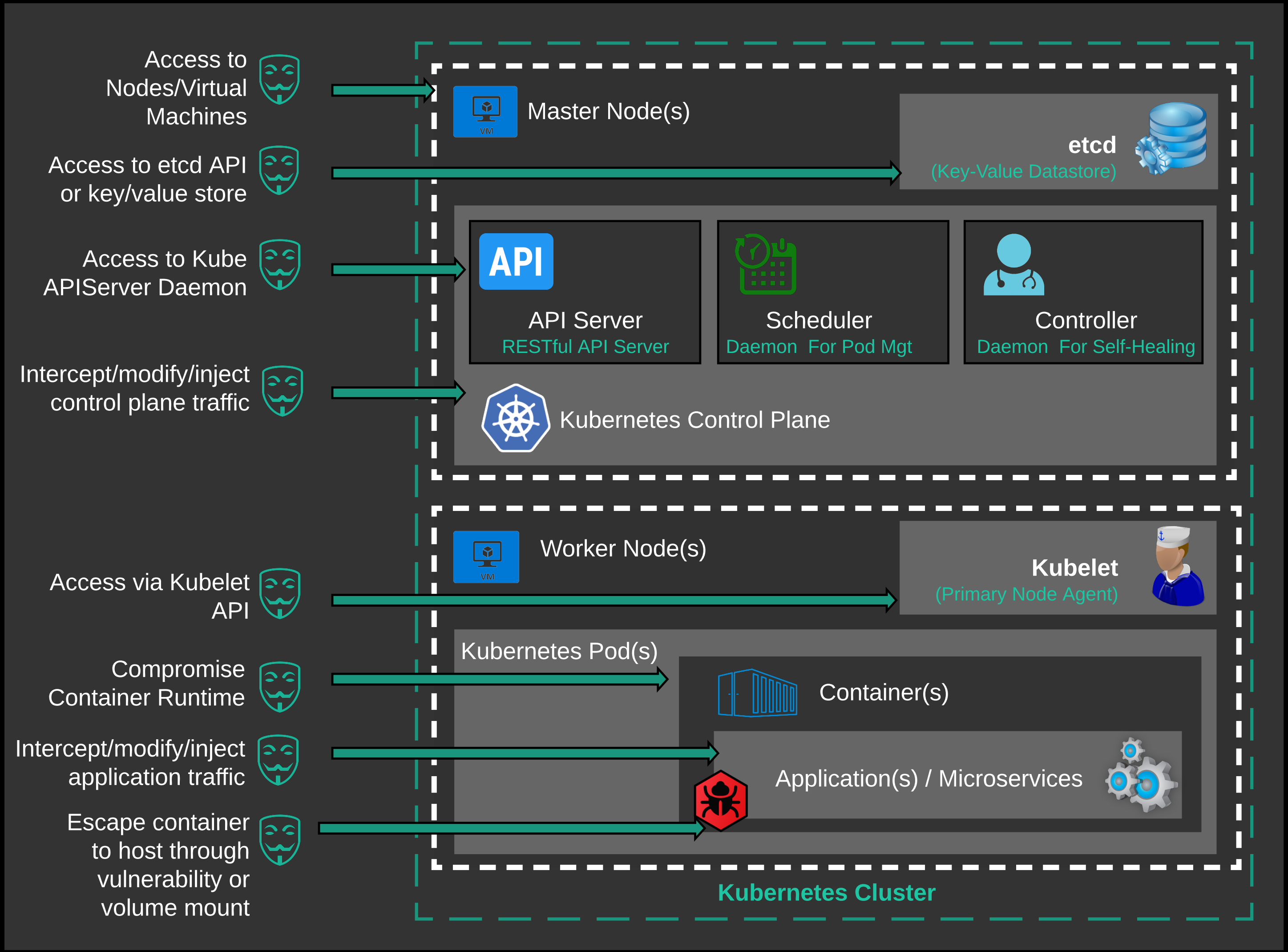
Kubernetes Security
Interactive Study Guide

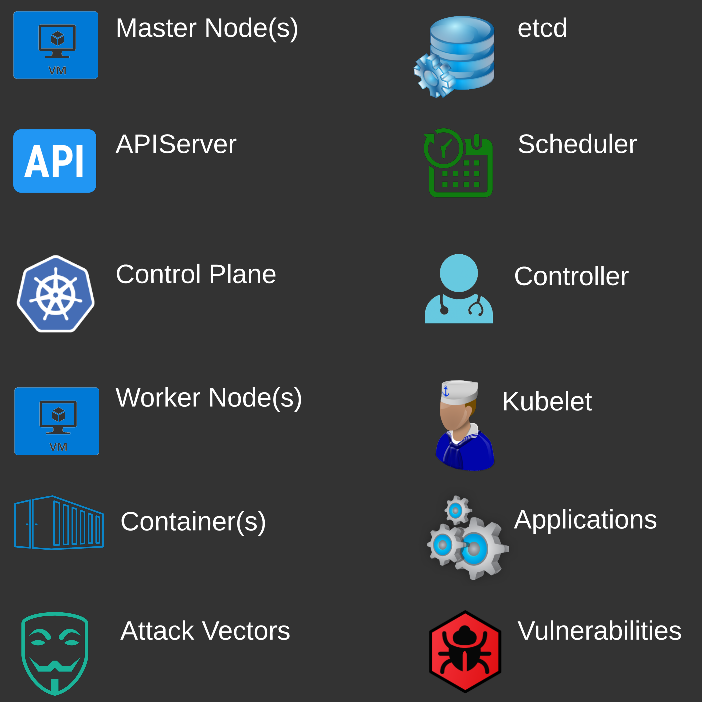
Click on any of the topics below to navigate this Study Guide

Kubernetes Architecture



Kubernetes Architecture and Attack Vectors

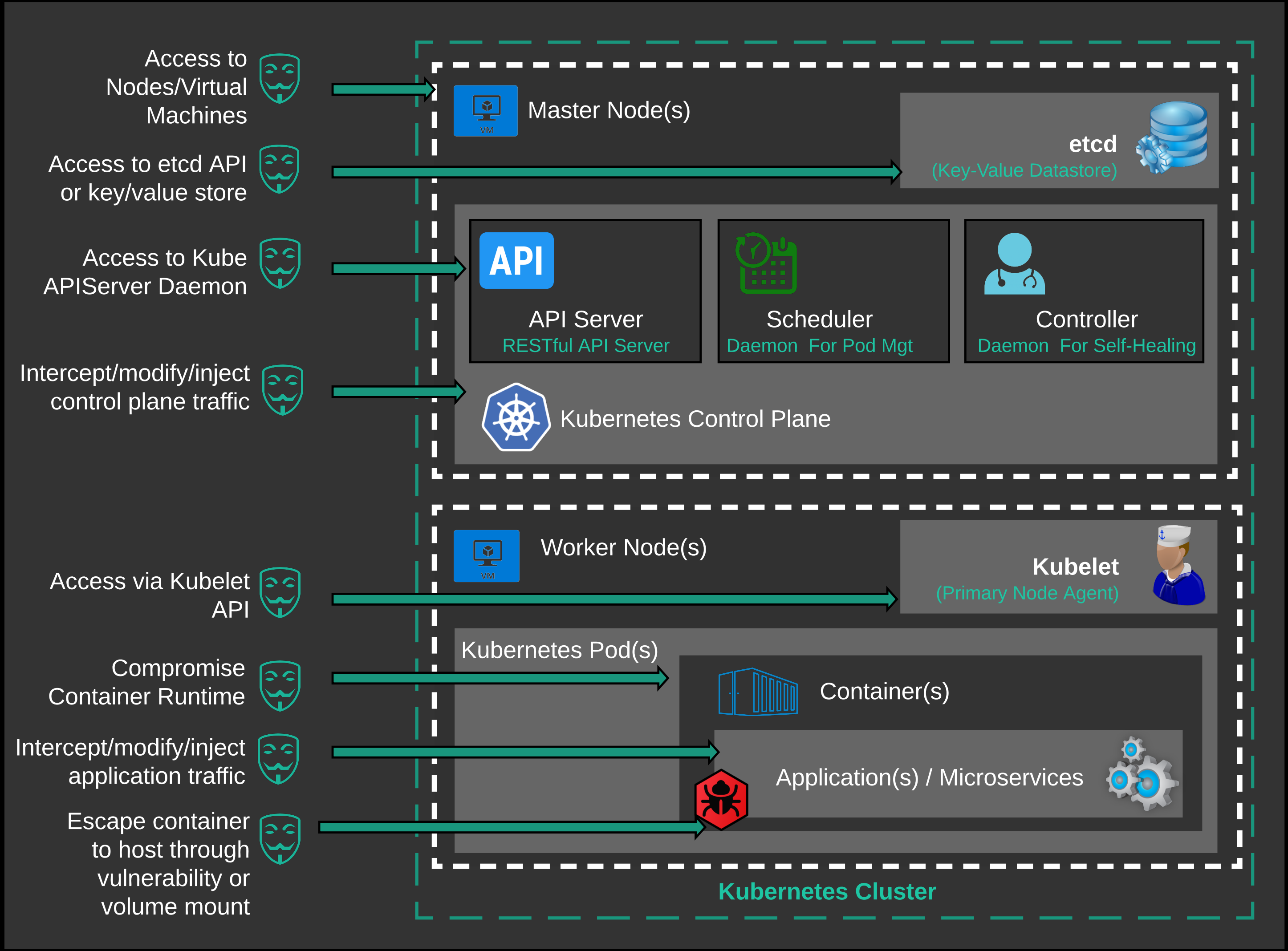
The diagram at the left is an interactive diagram to guide the student through the Kubernetes architecture and the attack vectors often exploited by cyber attacks. The following icons in the diagram are clickable on this guide to learn more.



Linux Academy

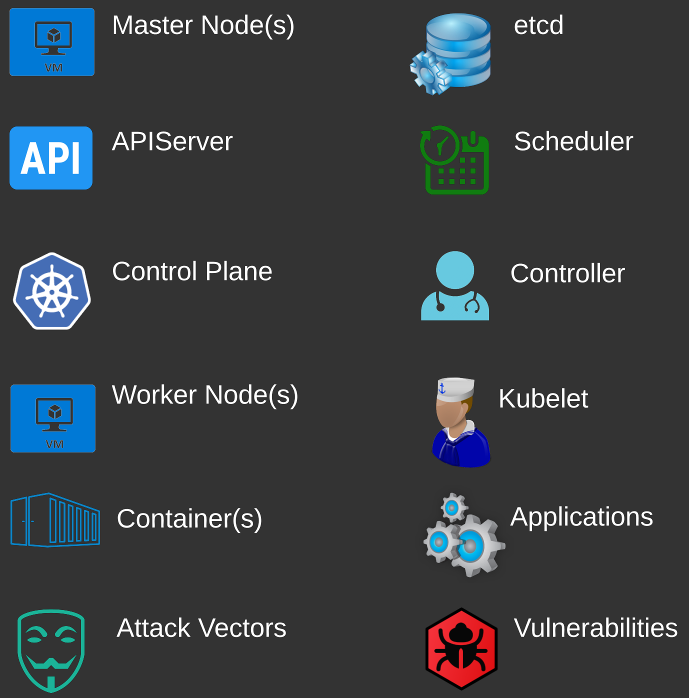
Kubernetes Security

Kubernetes Architecture and Attack Vectors



Kubernetes Architecture and Attack Vectors

The diagram at the left is an interactive diagram to guide the student through the Kubernetes architecture and the attack vectors often exploited by cyber attacks. The following icons in the diagram are clickable on this guide to learn more.

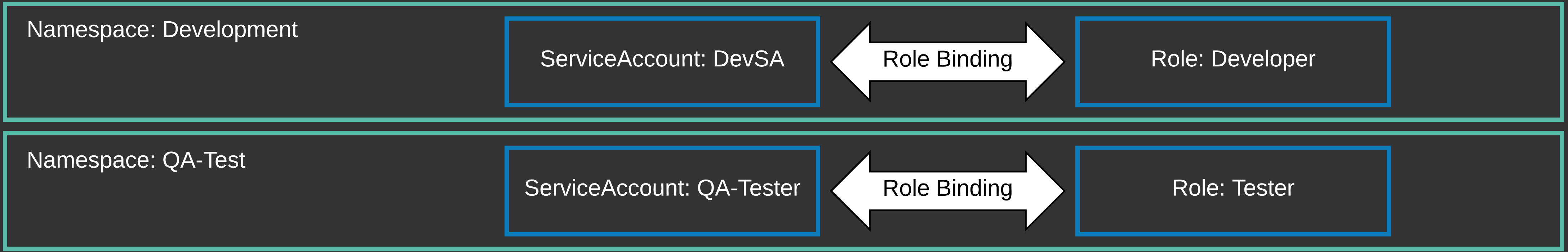


Linux Academy
Kubernetes Security

Principle of Least Privilege

"In information security, computer science, and other fields, the **principle of least privilege** (**PoLP**, also known as the **principle of minimal privilege** or the **principle of least authority**) requires that in a particular abstraction layer of a computing environment, every module (such as a process, a user, or a program, depending on the subject) must be able to access only the information and resources that are necessary for its legitimate purpose." [1]

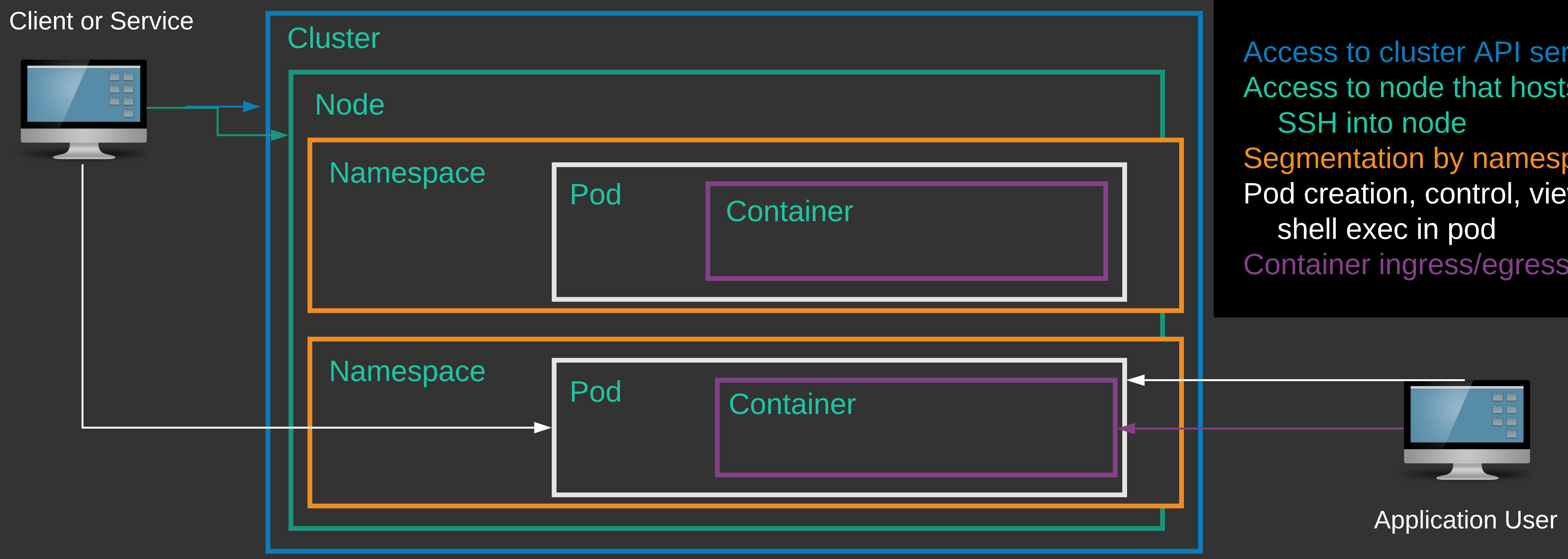
[1] https://en.wikipedia.org/wiki/Principle_of_least_privilege



Namespace: Where can it be done? ServiceAccount: Specified in YAML Role: What can be done, by whom?

Security Boundaries

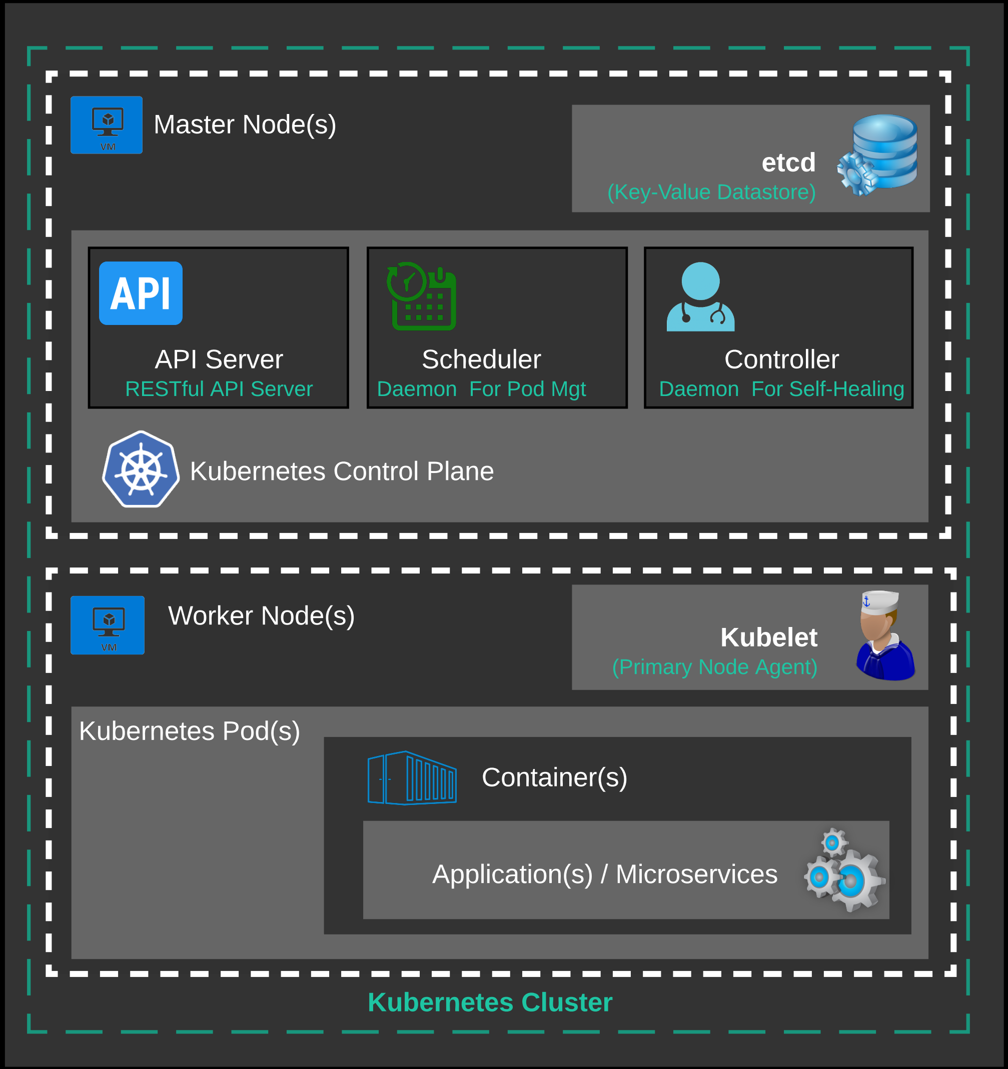
Default Security Boundaries in Kubernetes



Kubernetes Security Boundaries

Access to cluster API server
Access to node that hosts cluster
SSH into node
Segmentation by namespace
Pod creation, control, view
shell exec in pod
Container ingress/egress

TLS (Transport Layer Security)



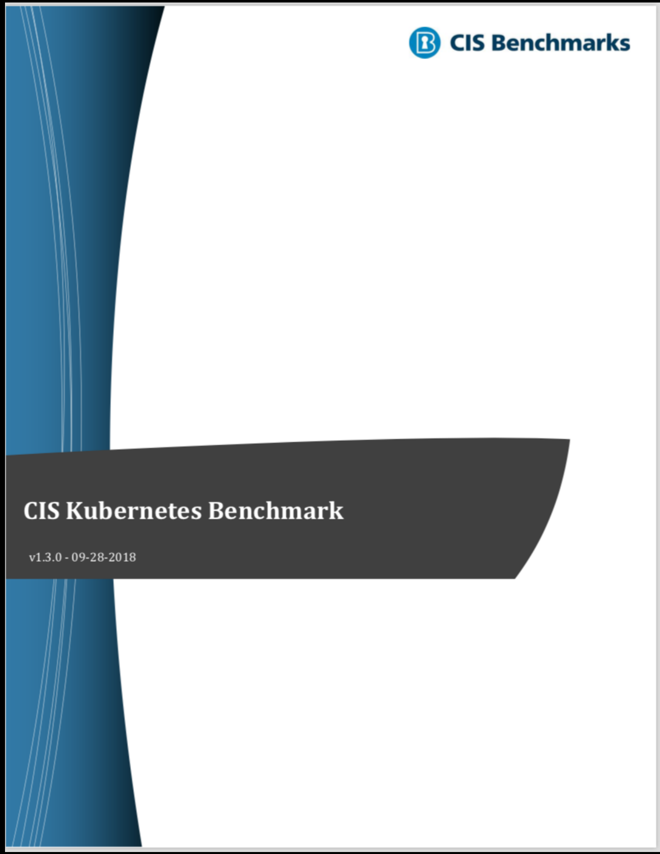
How Certificates are Used in a Cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

etcd also implements mutual TLS to authenticate clients and peers.

Using kube-bench to Harden a Cluster



kube-bench

Steps to Install and Run kube-bench

1) Use SSH to access your master node

```
$ ssh cloud_user@<Your IP Here>
```

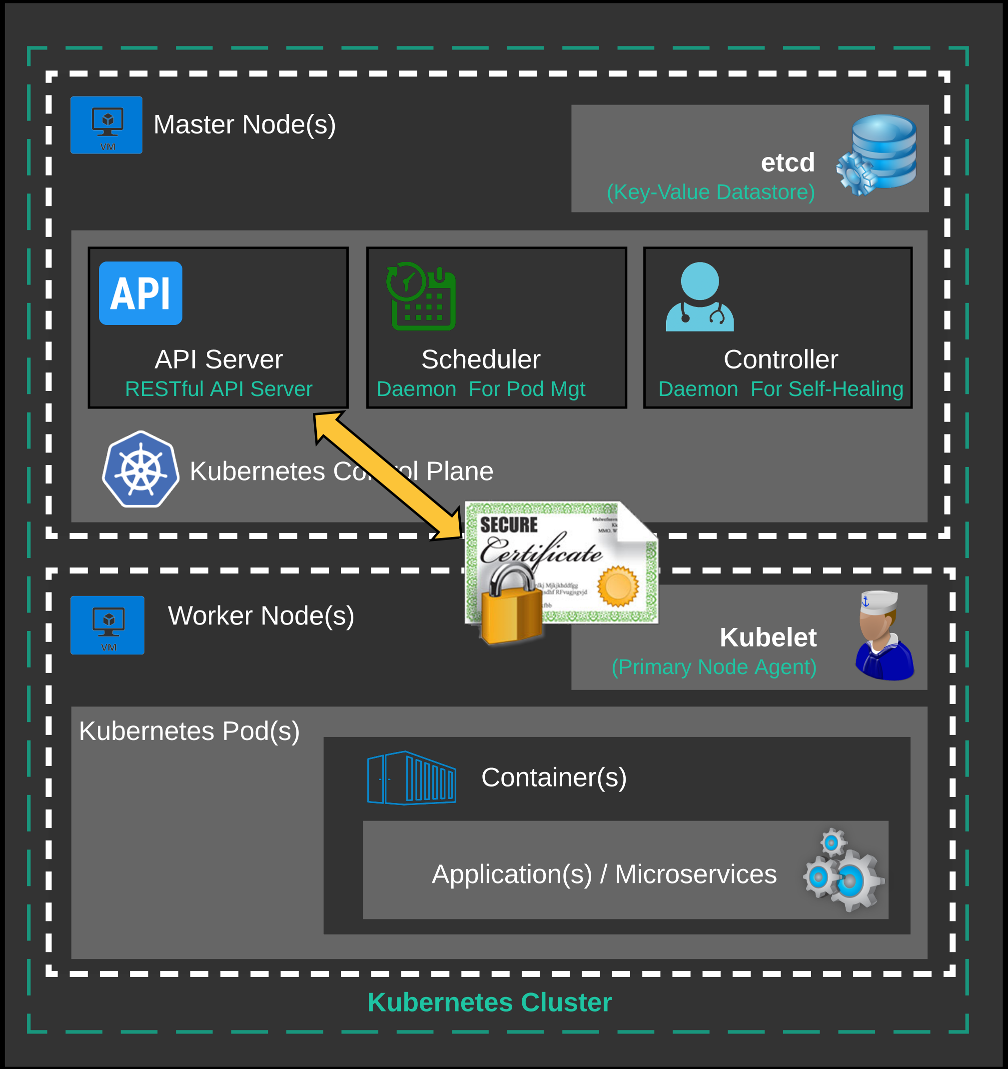
2) Run Docker to install from a container image

```
$ docker run --rm -v `pwd`:/host aquasec/kube-bench:latest install
```

3) Execute the kube-bench utility

```
$ ./kube-bench master
```

Securing the Kubelet



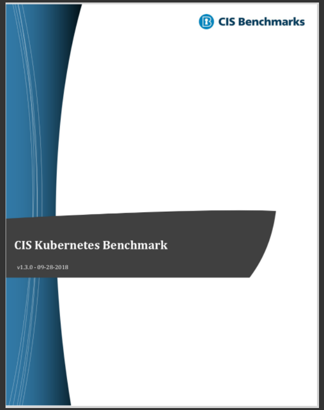
CIS Benchmark Recommendations on Kubelet Configuration

The following are a few examples of the command line arguments that should be reviewed as part of the kubelet hardening and configuration process:

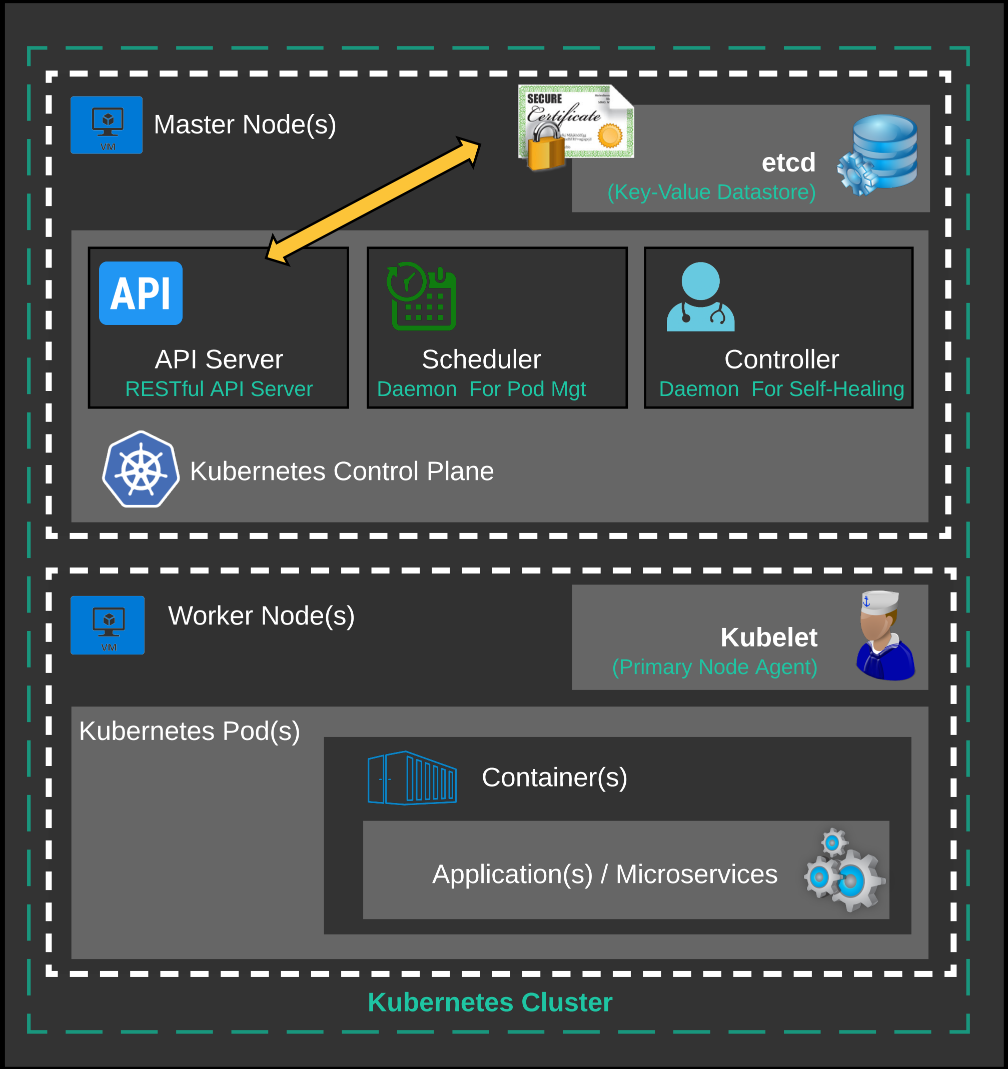
--allow-privileged: Set to false

(v1.11 and later, set this to true and recommend PodSecurityPolicy settings to prevent privileged containers)

--anonymous-auth: Set to false
--authorization-mode: Avoid AlwaysAllow setting
--client-ca-file: Should be set to valid certificates
--read-only-port: Set to 0 and readOnlyPort specified in Kubelet config
--streaming-connection-idle-timeout: Set to prevent denial of service attacks
--protect-kernel-defaults: Set to true
...
--tls-cert-file: Set as appropriate



Securing the etcd Key-Value Datastore



CIS Benchmark Recommendations on etcd

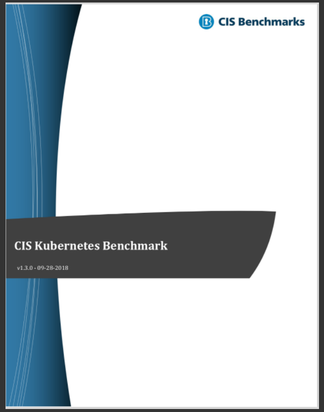
The following are a few examples of the command line arguments that should be reviewed as part of the etcd hardening and configuration process:

- --etcd-certfile and --etcd-keyfile: Should be set
- --enable-admission-plugins: Set to include a value for ServiceAccount
- --tls-cert-file and --tls-private-key-file: Should be set
- --auto-tls: Should be set to false
- --etcd-ca-file: Should be set to valid certificate
- --etcd-cafile on APIServer should be set to CA that signed etcd certificates

Most Kubernetes enterprise installations run etcd on a separate node, and for HA (High-Availability), more than one etcd server is configured for redundancy.

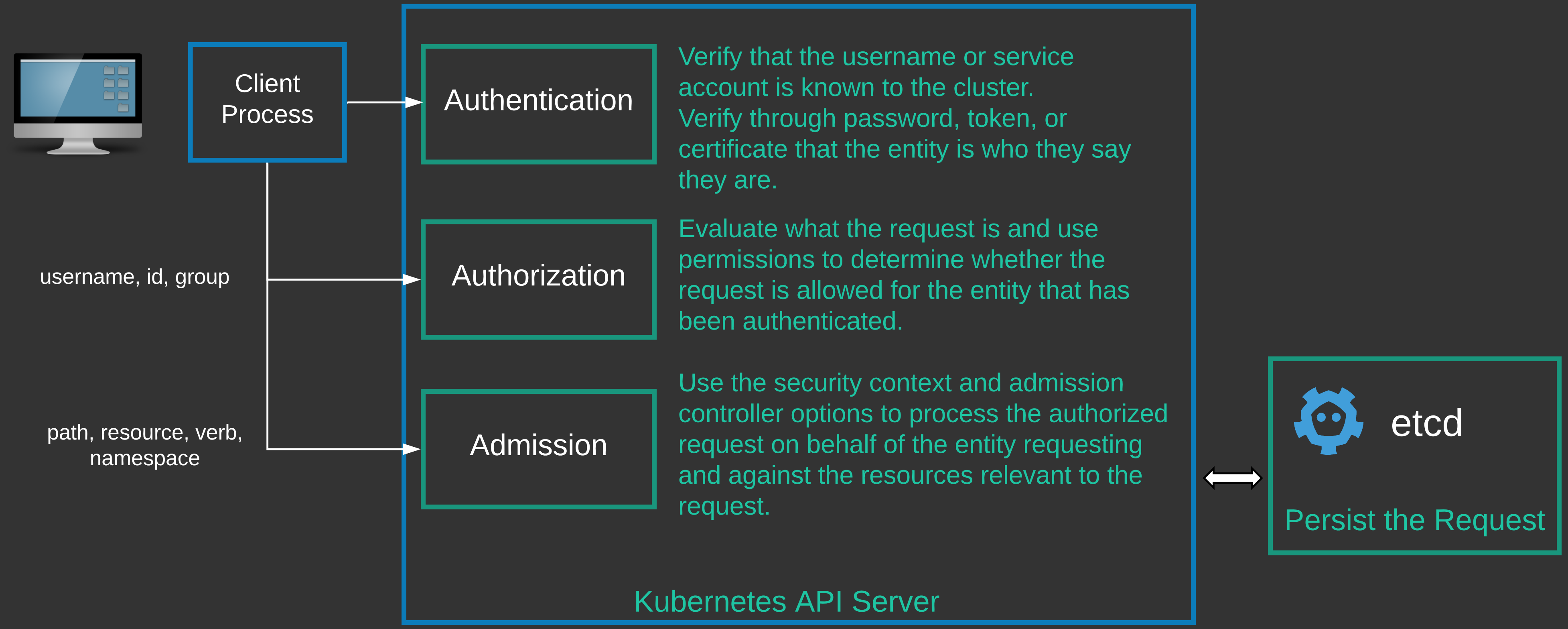
If multiple etcd nodes:

- --peer-client-cert-auth: Set to True
- --peer-auto-tls: Set to False
- --peer-cert-file and --peer-key-file: Set to Certificates
- --peer-trusted-ca-file



Authentication, Authorization, and Admission

The Authentication, Authorization, and Admission Control Process



Working with ServiceAccount Tokens

The following command may be used to create a ServiceAccount:

```
$ kubectl create serviceaccount jenkins
serviceaccount/jenkins created
```

The created secret holds the public CA of the API server and a signed JSON Web Token (JWT)

To display the YAML revealing the associated secret:

```
$ kubectl get serviceaccounts jenkins -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: "2019-02-04T19:28:48Z"
  name: jenkins
  namespace: default
  resourceVersion: "112679"
  selfLink: /api/v1/namespaces/default/serviceaccounts/jenkins
  uid: 183f9cbd-28b3-11e9-af90-062d4745d730
secrets:
- name: jenkins-token-mgtnp
```

The command to display available tokens is:

```
$ kubectl get secrets
NAME                 TYPE                                  DATA AGE
default-token-l4w8h  kubernetes.io/service-account-token   3    4d21h
jenkins-token-mgtnp  kubernetes.io/service-account-token   3    20m
```

Identity

- Kubernetes doesn't have a notion of a human user.
- Kubernetes assumes that 'users' are managed outside of Kubernetes.
- In production environments, organizations use technologies such as SSO (Single Sign-On), LDAP (Lightweight Directory Access Protocol), SAML (Security Assertion Markup Language) and Kerberos for identity management.
- Outside of production, in development, lab, or test environments, other 'Authentication Strategies' may be employed.

Authorization Modes

Kubernetes supports the following authorization modes:

Node Authorization: A special-purpose authorizer that grants permissions to kubelets based on the pods they are scheduled to run on.

Attribute-Based Access Control: An authorizer through which access rights are granted to users through policies combining attributes (user attributes, resource attributes, objects, etc.)

Webhook: A webhook is an HTTP callback—an HTTP POST that occurs when something happens. This mode allows for integration with Kubernetes external authorizers.

Role-Based Access Control: A method of regulating access to computer or network resources based on the roles of individual users within an enterprise.

Admission Control

Kubernetes supports over 30 admission controllers. Subsequent to authentication and authorization, admission controllers are the final step in a three-step process before Kubernetes persists the resource in etcd. Some relevant admission controllers to ensure running containers securely are:

AlwaysPullImages: While there is a performance advantage to storing and reusing images on a node, hygiene and the assurance that you always run up-to-date container images may be important. Since vulnerabilities are patched upstream, pulling images ensures that the latest remediations are always downloaded.

DenyEscalatingExec: When hackers open shells in privileged containers, they have access to the host. This option ensures that exec and attach commands from privileged containers are blocked.

PodSecurityPolicy: This option implements pod admission based on security context and available policies.

LimitRange and **ResourceQuota**: To prevent denial of service attacks, and any spawning of unauthorized processes from established pods, this option observes incoming requests for violation of these limits.

NodeRestriction: This limits the permissions of each kubelet, ensuring that it can only modify pods that are bound to it and its own Node object.

Security Context

Implement Discretionary Access Control
Limit access based on user or group ID

Capabilities
Confine root access to certain commands

Apply Profiles
Configure seccomp or use AppArmor to restrict system calls made from processes

Implement Mandatory Access Control
Use SELinux to assign security labels to operating system objects.

Example: Use **runAsUser** and **allowPrivilegeEscalation** to limit a pod's permissions

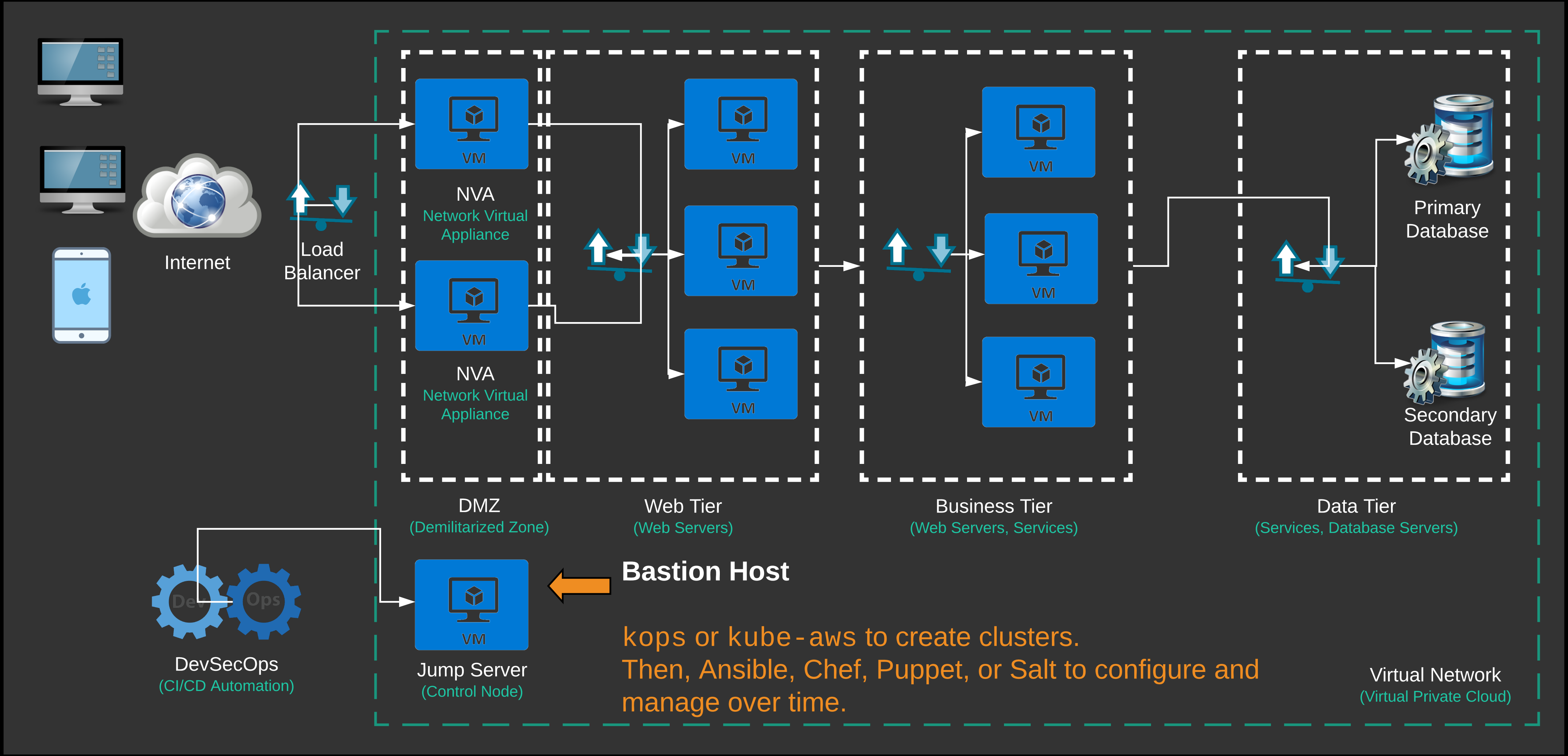
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

What is a Pod Security Policy?

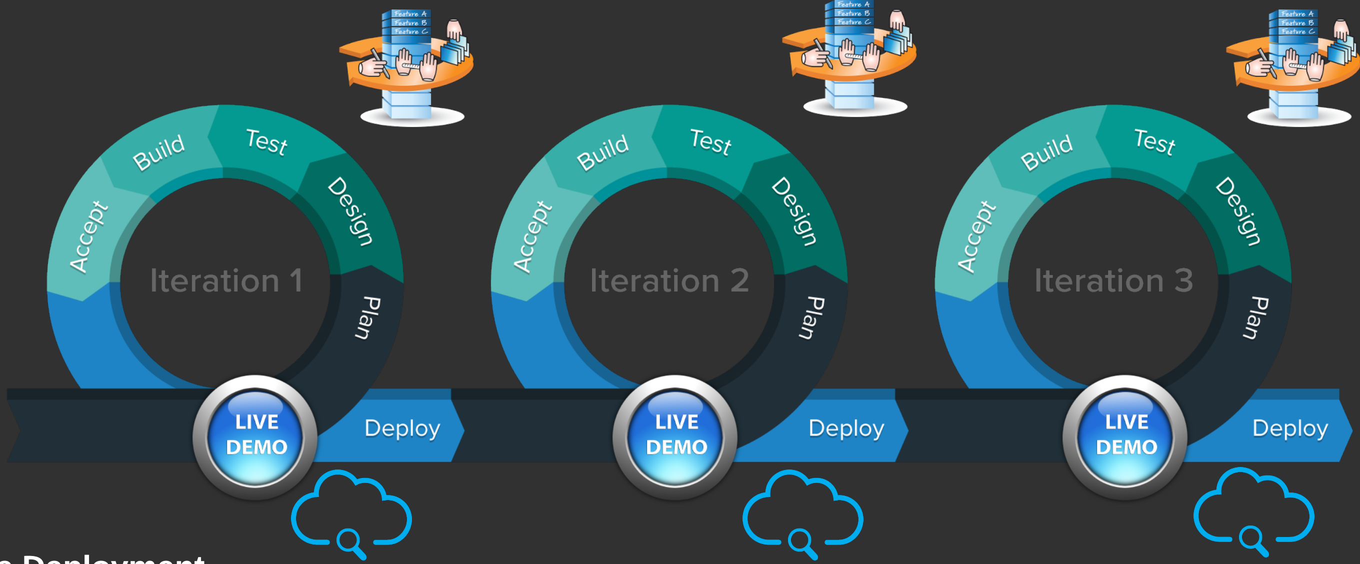
A *Pod Security Policy* is a cluster-level resource that controls security sensitive aspects of the pod specification. The **PodSecurityPolicy** objects define a set of conditions that a pod must run with in order to be accepted into the system, as well as defaults for the related fields. They allow an administrator to control the following:

Control Aspect	Field
Running of privileged containers	**privileged**
Usage of host namespaces	**hostPID**, **hostIPC**
Usage of host networking and ports	**hostNetwork**, **hostPorts**
Usage of volume types	**volumes**
Usage of the host file system	**allowedHostPaths**
Whitelist of Flexvolume drivers	**allowedFlexVolumes**
Allocating an FSGroup that owns the pod's volumes	**fsGroup**
Requiring the use of a read-only root file system	**readOnlyRootFilesystem**
The user and group IDs of the container	**runAsUser**, **runAsGroup**, **supplementalGroups**
Restricting escalation to root privileges	**allowPrivilegeEscalation**, **defaultAllowPrivilegeEscalation**
Linux capabilities	**defaultAddCapabilities**, **requiredDropCapabilities**, **allowedCapabilities**
The SELinux context of the container	**seLinux**
The Allowed Proc Mount types for the container	**allowedProcMountTypes**
The AppArmor profile used by containers	annotations
The seccomp profile used by containers	annotations
The sysctl profile used by containers	annotations

Immutable Architecture: Using a Bastion Host



Agile Process

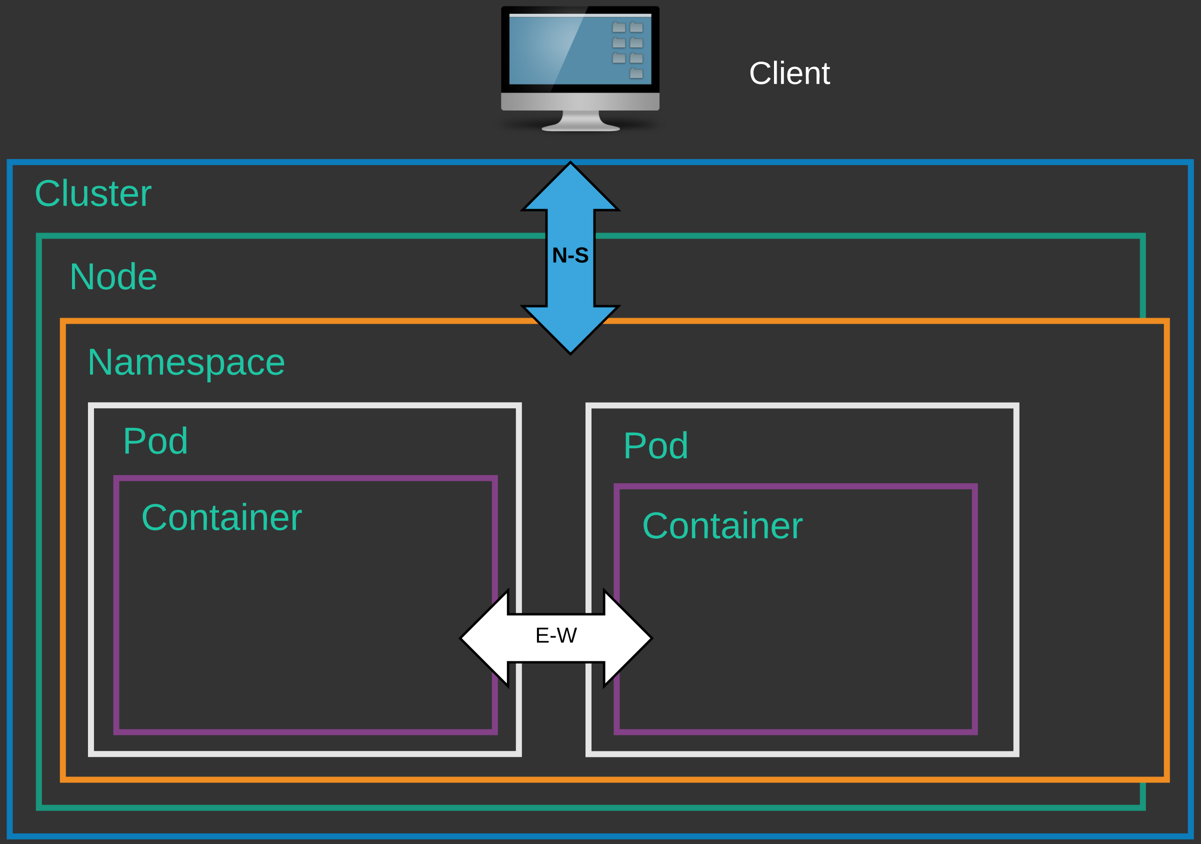


Iterative Deployment

Agile 'iterative' processes implement 'Continuous Delivery' and thus mandate the need for automated deployments performed on demand.

Agile	DevSecOps	Forrester	Beyond Kube

Limiting Pod-to-Pod Traffic



Ingress and Egress

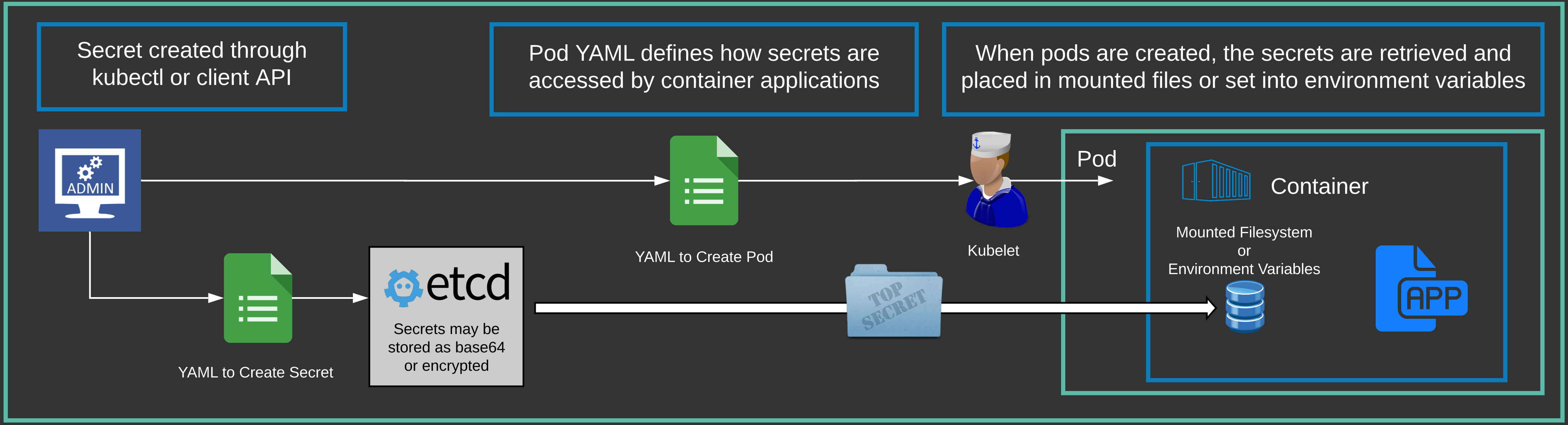
North-South Traffic
Traffic from outside the cluster.

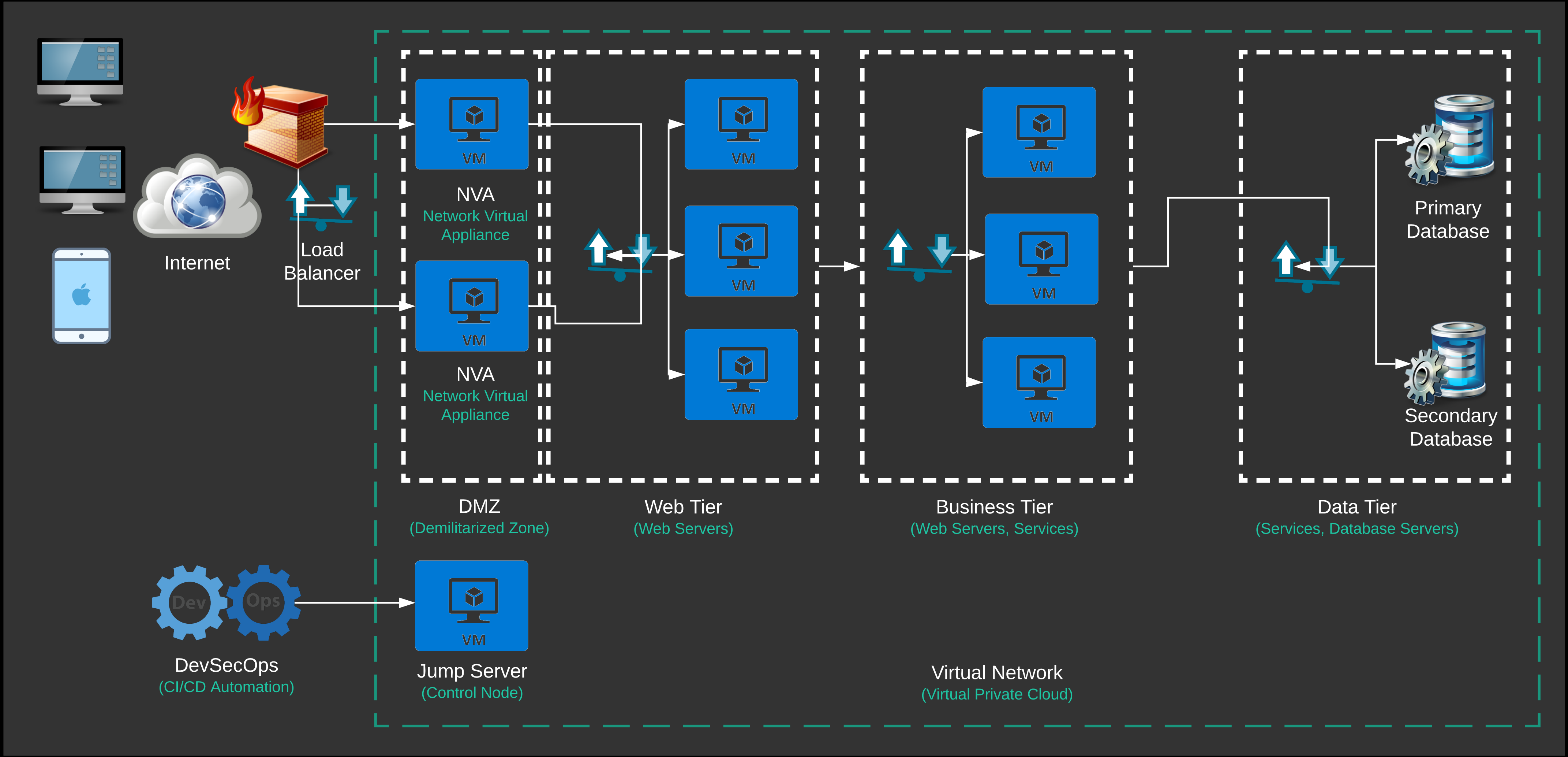
East-West Traffic
Intra-Cluster traffic, often pod-to-pod traffic for peer-to-peer communications.

Managing Secrets

Secrets, such as username/password, tokens, RSA keys, and other authentication credentials, are required by many containerized applications that run in pods. Kubernetes offers a resource 'secret' that provides a means of making secrets available to applications initiated by a deployment, ReplicaSet, or other form of pod creation. Secrets may be stored in etcd or third-party systems. They are passed to applications via mountable files systems or as environment variables.

Secrets Workflow





Linux Academy
Kubernetes Security

Firewalls and Virtual Private Networking

1 - 2 - 3 - 4 - 5