



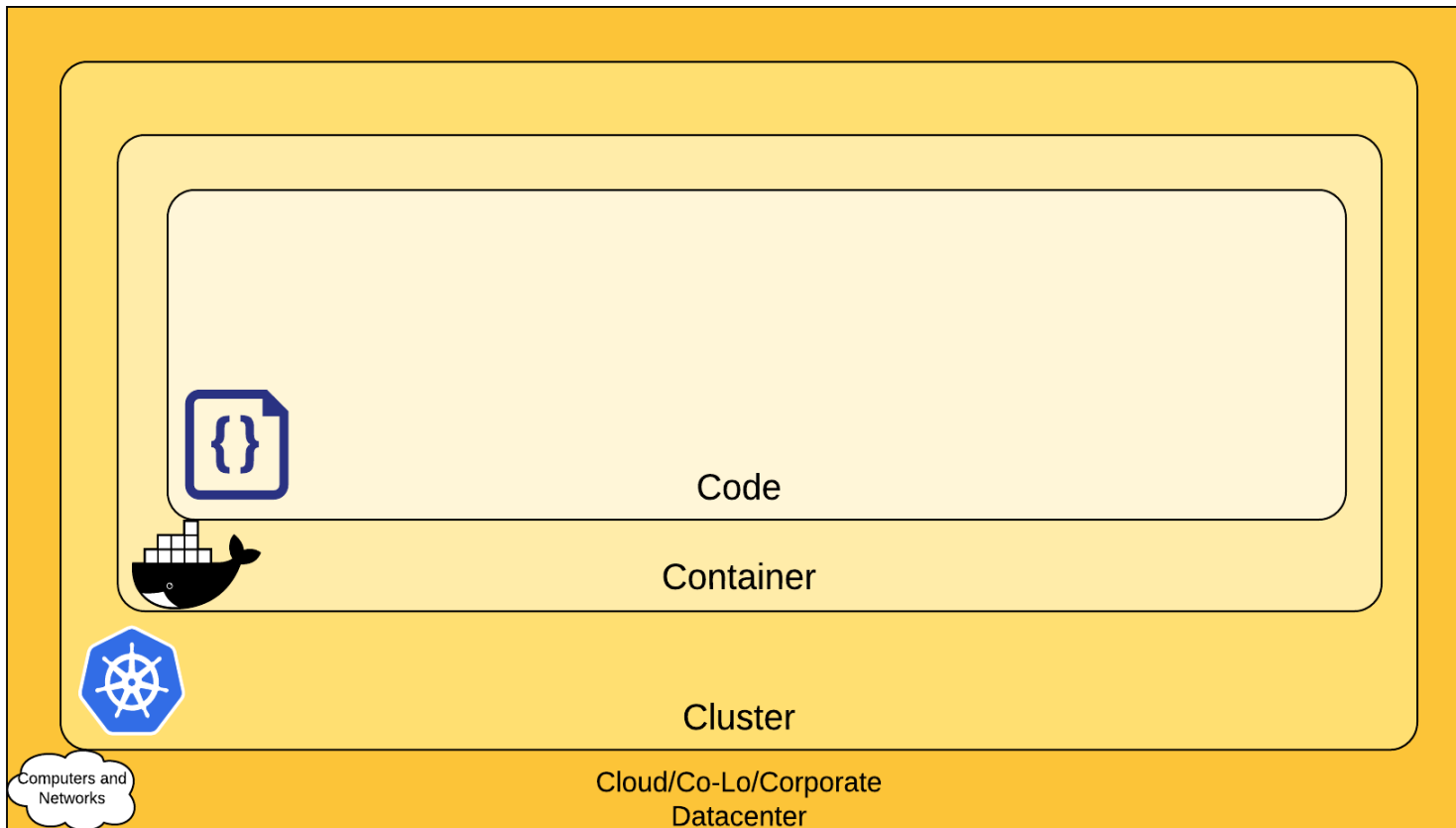
CONTAINER ISOLATION & LEAST PRIVILEGE

CORKSEC NOV 2020

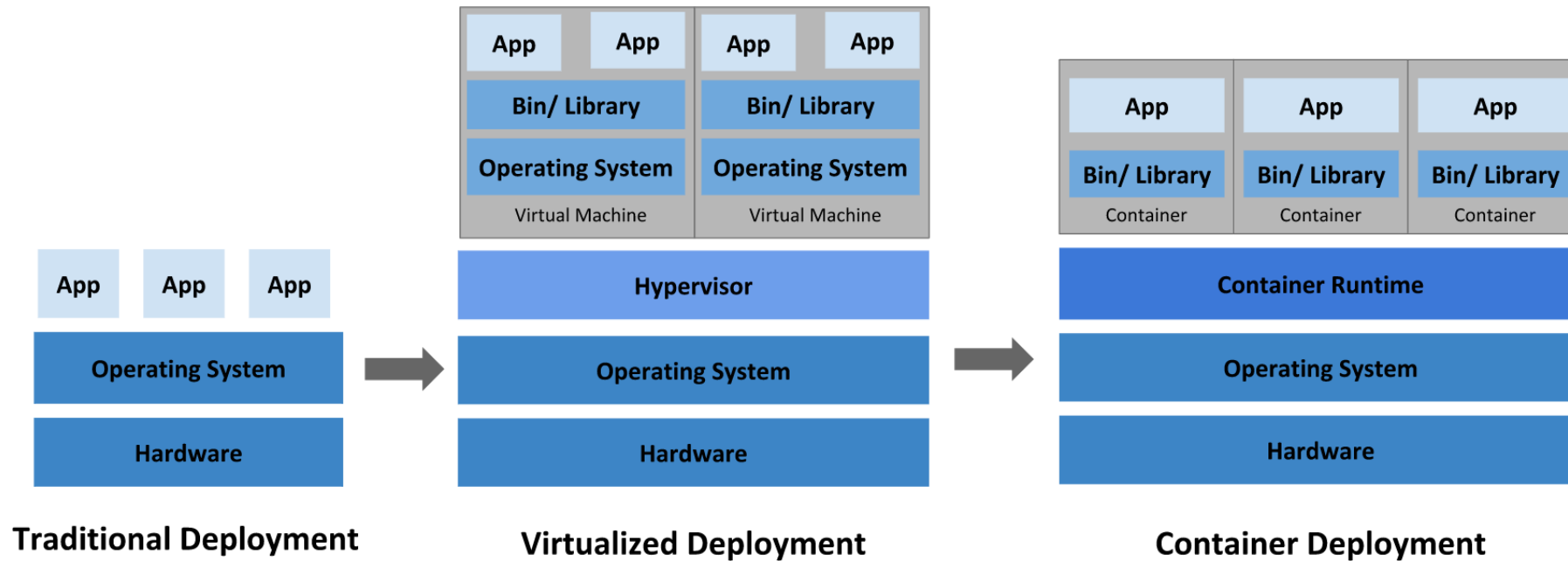
4 C's of Cloud Security

Exploring the Different Security Layers

Source: [Kubernetes](https://kubernetes.io/)



Containers 101



References: [#1](#), [#2](#)



Portable Runtime
Easily Deployed
Multi Cloud Adoption
Less OS Licensing
Less OS Patching

A Docker container image is a lightweight, standalone, **executable package** of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

- The application runs quickly and reliably from one computing environment to another.
- These prepared runtimes are made available from container repositories such as DockerHub.

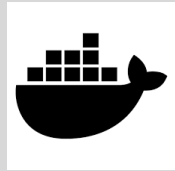
Building the Container

```
public class PingPong {  
  
    Run | Debug  
    public static void main(String[] args) throws Exception {  
        HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);  
        server.createContext("/ping", new MyHandler());  
        server.setExecutor(null);  
        server.start();  
    }  
  
    static class MyHandler implements HttpHandler {  
        @Override  
        public void handle(HttpExchange t) throws IOException {  
            String response = "pong\n";  
            t.sendResponseHeaders(200, response.length());  
            OutputStream os = t.getResponseBody();  
            os.write(response.getBytes());  
            os.close();  
        }  
    }  
}
```

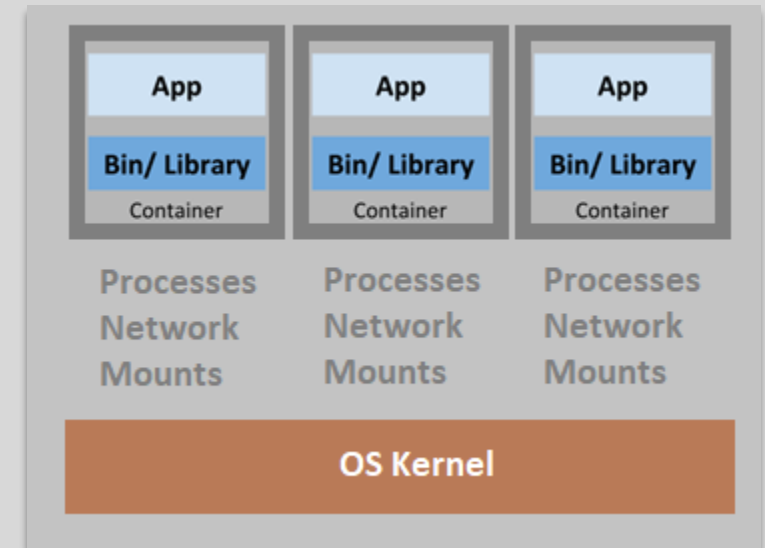
```
 Dockerfile >  FROM  
1 FROM java:8  
2 COPY PingPong.java /  
3 RUN javac PingPong.java  
4 EXPOSE 8080  
5 ENTRYPOINT ["java"]  
6 CMD ["PingPong"]
```

- https://hub.docker.com/_/openjdk
- Oracle Linux based image
- Includes Package Managers & Shell

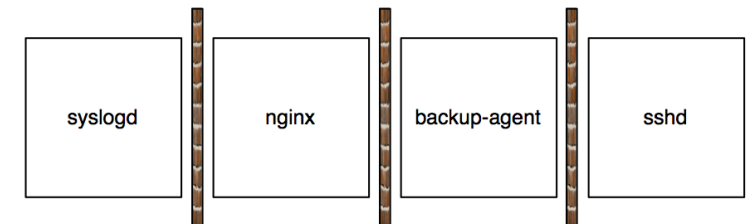
Docker Runtime



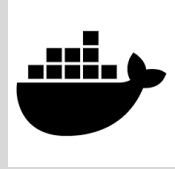
- [Docker Security](#) When you start a container with `docker run`, behind the scenes Docker creates a set of **namespaces** and **control groups** for the container.
 - **processes** running within a container cannot see, or affect processes running in another container, or in the host system.
 - Each container gets its own **network stack** - container doesn't get privileged access to the sockets or interfaces of another container.
 - Control Groups implement resource accounting and limiting. to ensure that each container gets its fair share of memory, CPU, disk I/O;
- Can configure `/host` directory as the `/` directory on your container
 - Resources: Manipulating Mount features [F-Secure](#)
- Daemon requires root privileges (unless using Rootless mode). Docker is an unauthenticated API with privileged access to the kernel Docker
- **A root user within the container has root access outside the container.**
 - configure container to run as unprivileged user ([namespace remap](#) feature).
 - Specify user `docker run --user 1000 -v /:/home/notImportantDir/ innocent-docker-image`
- Containers [are like firewalls between processes](#)



Dockerized Host



Docker Runtime



- Linux Kernel Capabilities
 - Docker starts containers with a restricted set of capabilities - Fine Grained access control system.
 - Very infrequently would containers need “real” root privileges - containers can run with a reduced capability set
 - “root” within a container has much less privileges than the real “root”.
- For instance, it is possible to:
 - deny all “mount” operations;
 - deny access to raw sockets (to prevent packet spoofing);
 - deny access to some filesystem operations, like creating new device nodes, changing the owner of files, or altering attributes (including the immutable flag);
 - deny module loading;
- So if an intruder manages to escalate to root within a container, it is much harder to escalate to the host.
- By default Docker drops all capabilities except those needed, implements an allowlist instead of a denylist approach. You can see a full list of available capabilities in [Linux manpages](#).
- You can add an extra layer of safety by enabling AppArmor, SELinux, GRSEC ...

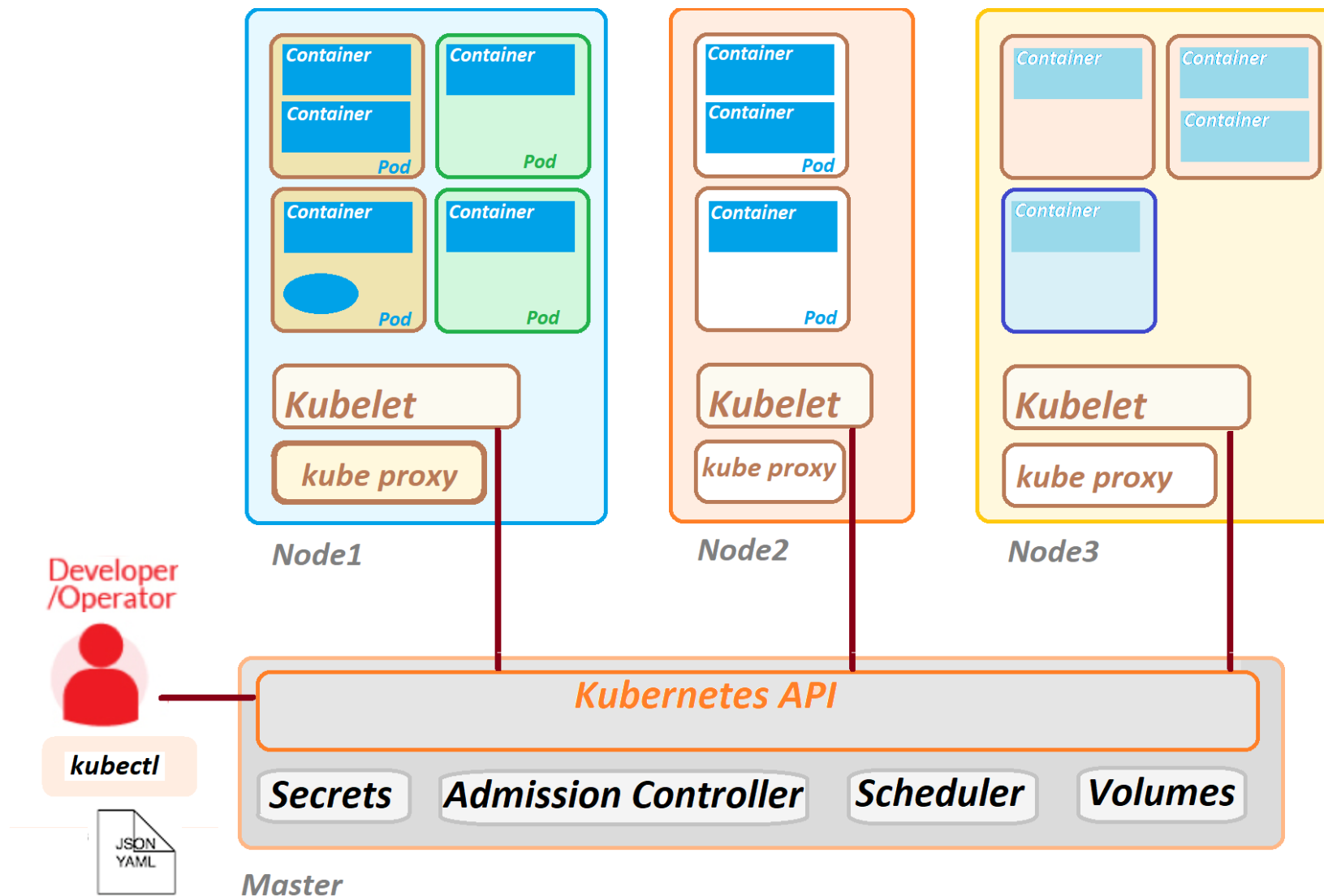
CIS Docker Benchmark

- <https://www.cisecurity.org/benchmark/docker>
- Host
 - Create a separate partition for containers
 - Harden host
 - Update kernel and Docker software
 - Only allow Trusted users to control the Docker Daemon
 - Audit Docker files and directories
- Docker Daemon configuration
 - Restrict network traffic between containers
 - Allow Docker to make changes to Iptables
 - Do not use aufs storage driver
 - Restrict ownership of Docker Daemon configuration files..
- Container Images
 - Create a user for the container
 - Use Trusted base images ...
 - Apply AppArmor or SELinux
 - Do not use Privileged containers
 - Restrict Linux kernel capabilities within container
 - Do not mount sensitive host system directories
 - Don't run SSH within containers
 - Don't map privileged ports
 - Don't use host network mode
 - Mount container root file system as read only
 - Do not share the host processes namespace

Kubernetes (K8S) 101



- Platform for automating deployment & operations of application containers across clusters of hosts.
- Automated **Scheduling, Self-Healing, rollouts & rollback, Horizontal Scaling & Load Balancing**
- Designed by Google, now maintained by the Cloud Native Computing Foundation. v1 was in 2015
- Open Source. Provides a higher density of resource utilization. Create Ephemeral, disposable Containers
- Kubernetes supports *Docker* containers since its first version. In July 2016 *rkt* container engine was added.
- **Node:** may be a virtual or physical machine, depending on the cluster. Each node contains the services necessary to run Pods, managed by the control plane.
- **Pod:** When you create a deployment Kubernetes creates a Pod to host your application instance. It represents a unit of deployment
- A Pod encapsulates an application's container (or multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run.
- A pod containers are guaranteed to be co-located on the host machine and can share resources.



Controllers continuously check the cluster's current state against the desired state of the cluster. If there are any differences, controllers perform tasks to make the current state match the desired state.



Kubernetes API

Kubectl CLI admin tool to get info about deployed applications

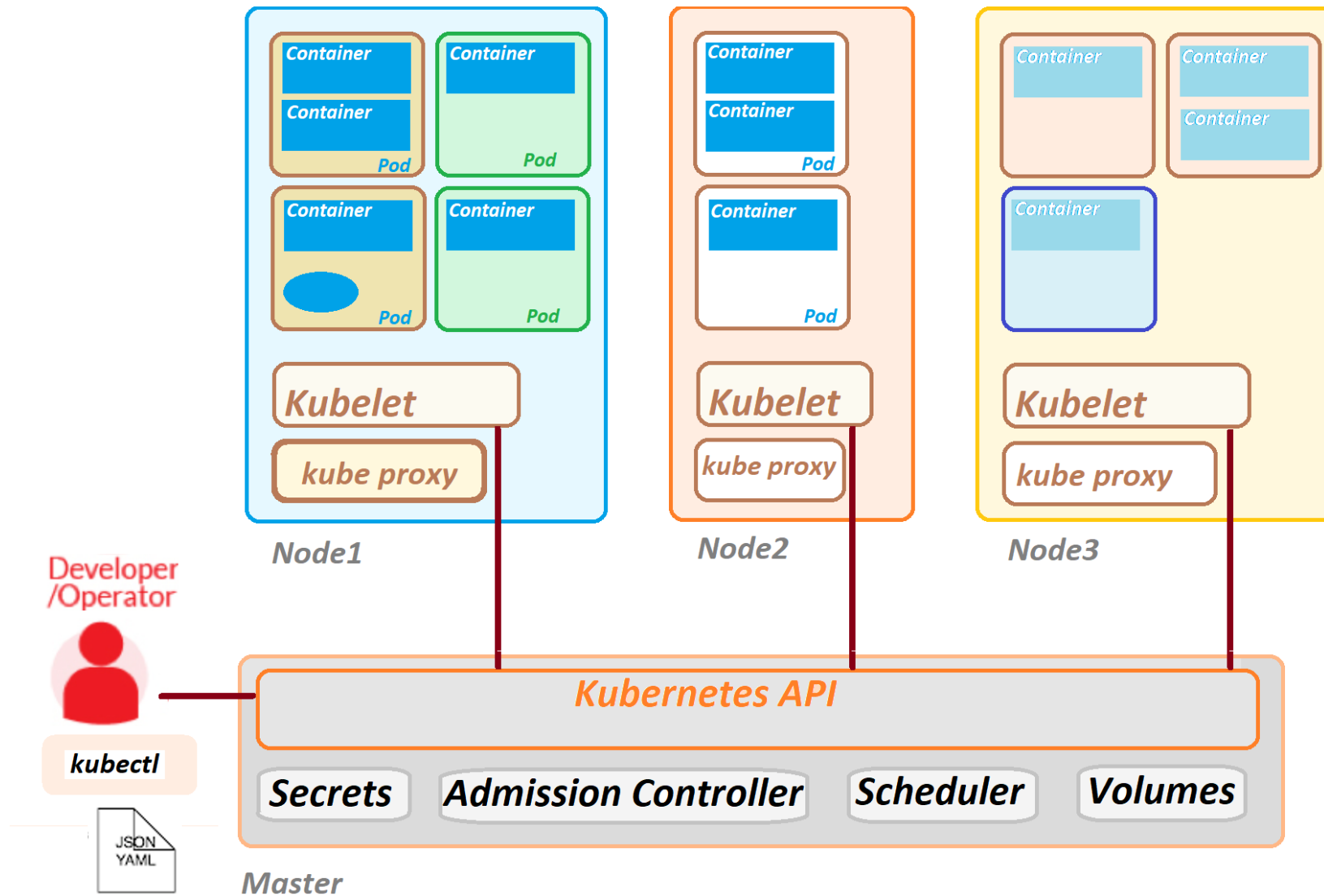
- kubectl get - list resources
- kubectl describe - show detailed resource info
- kubectl logs - logs from a container in a pod
- kubectl exec - execute a command on a container in a pod

Kubernetes API – entry point for all administrative tasks.

There might be more than one master node in the cluster

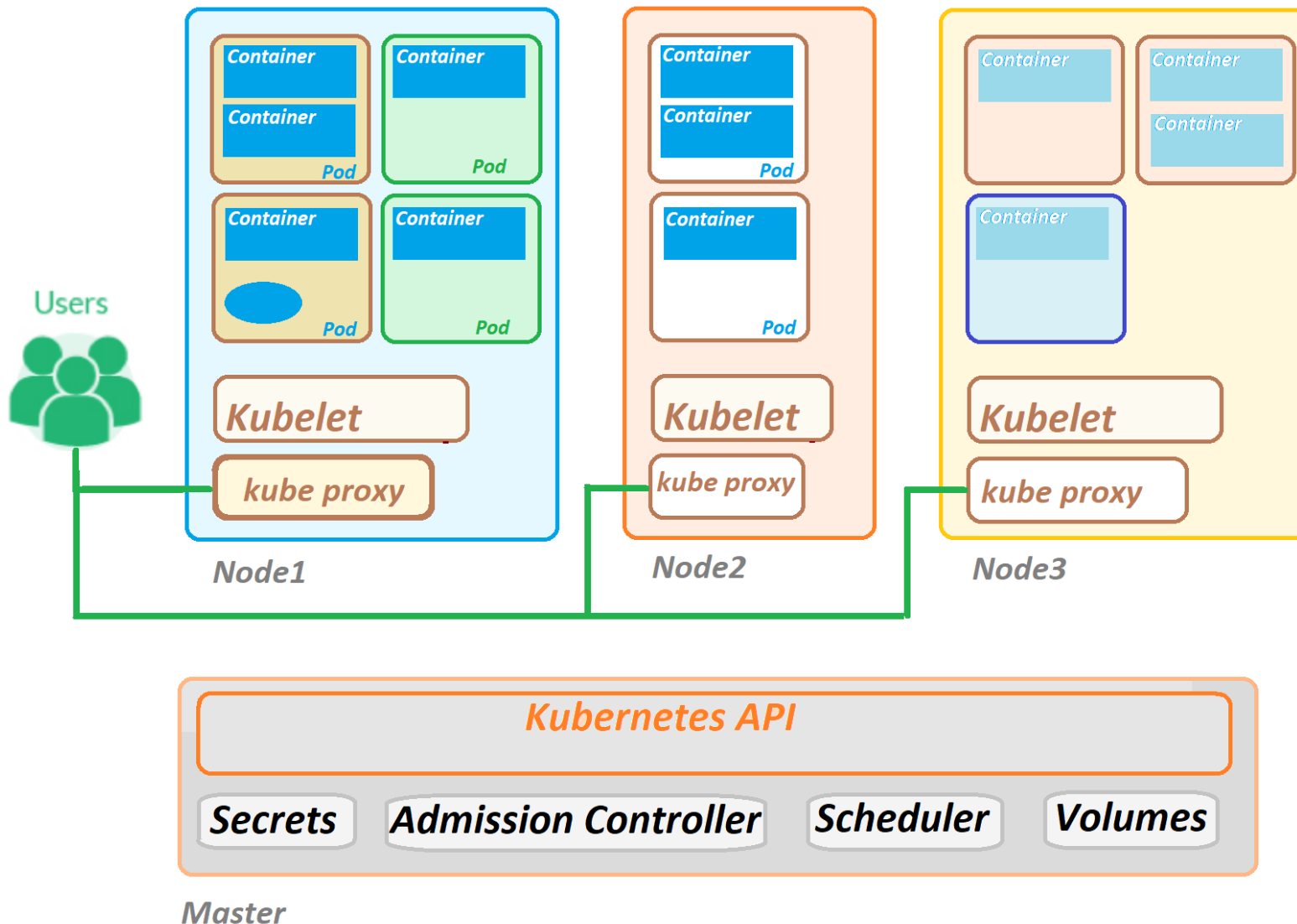
Scheduler - schedules tasks to the nodes

References: [#1](#), [#2](#)



Kubelet

- ensure all containers on the node are healthy
- takes care of starting / stopping / maintaining application containers organized into pods as directed by the control plane



Kube proxy

an implementation of a network proxy & load balancer

supports the **service** abstraction along with other networking operations.

Routes traffic to the appropriate container based on IP and port of the incoming request.

[Ingress Controllers](#)

A **Service** definition exposes an application running on a set of Pods as a network service. Gives Pods their own IP and a single DNS name for a set of Pods. can load-balance across them.

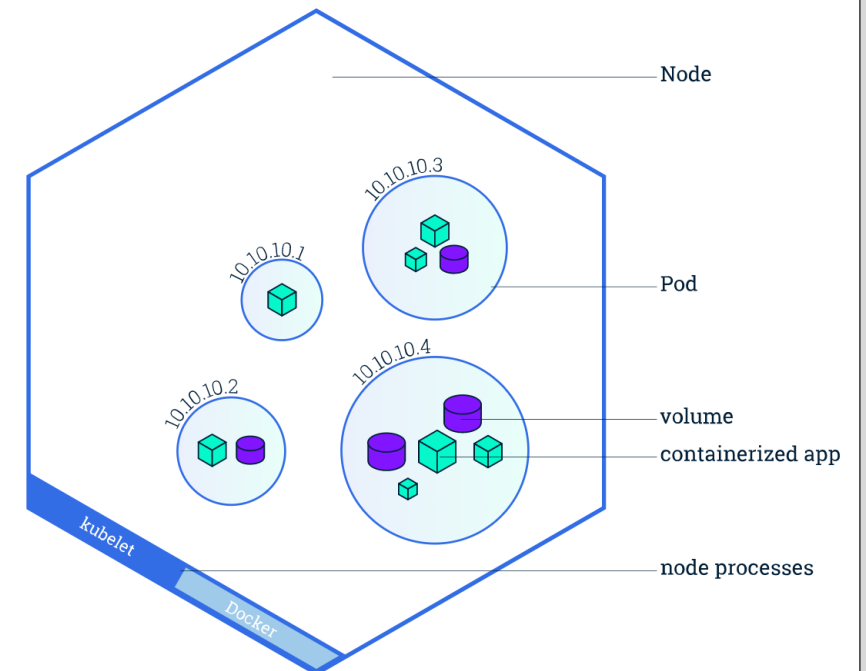
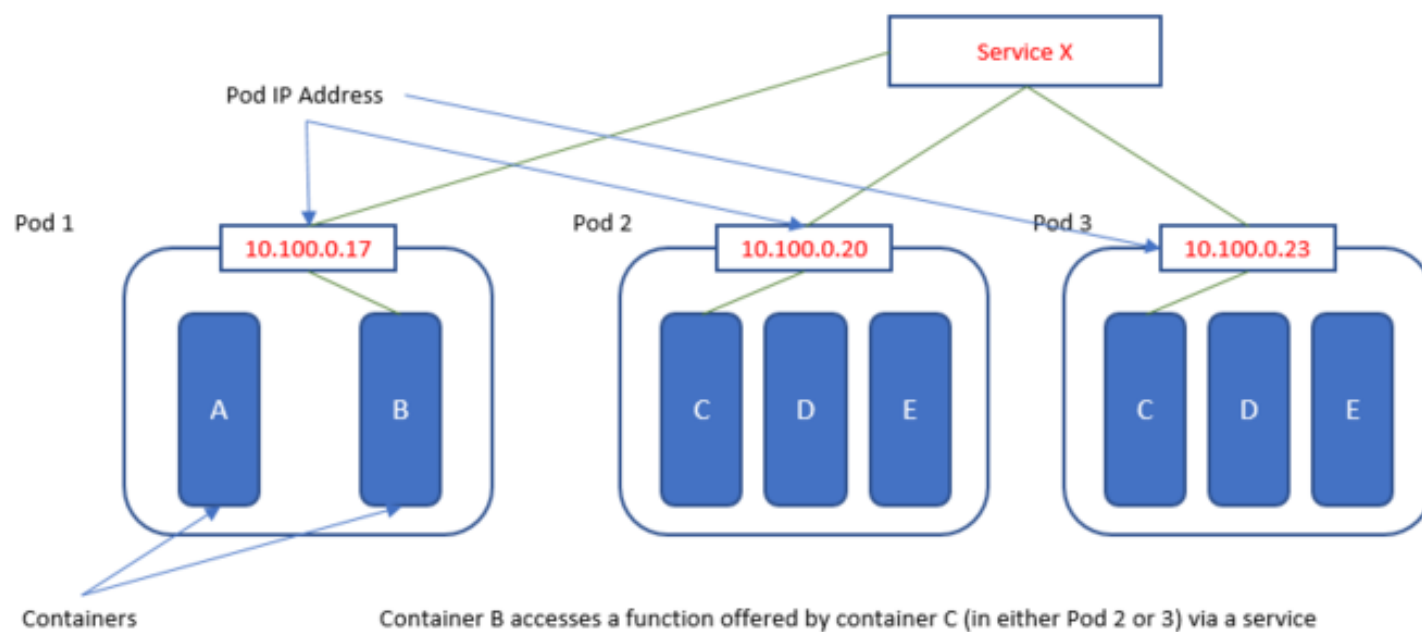
Kubernetes Networking

- Every Pod gets its own IP address
- IPs are typically non-routable outside of the K8S cluster.
- Containers within a Pod share their IP address.
- By Default Pods on a node can communicate with all pods of all nodes without NAT – flat network
- IP Addresses are ephemeral, Pod Locations are non-deterministic
- Lots of different Kubernetes Container Network Interface (CNI) Implementations [available](#).
- Can define [Network Policies](#) to specify what traffic to allow to and from Pods
 - implemented by Network plugins; By default, pods are non-isolated & accept traffic from any source
 - **To isolate Pods** define a NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy.





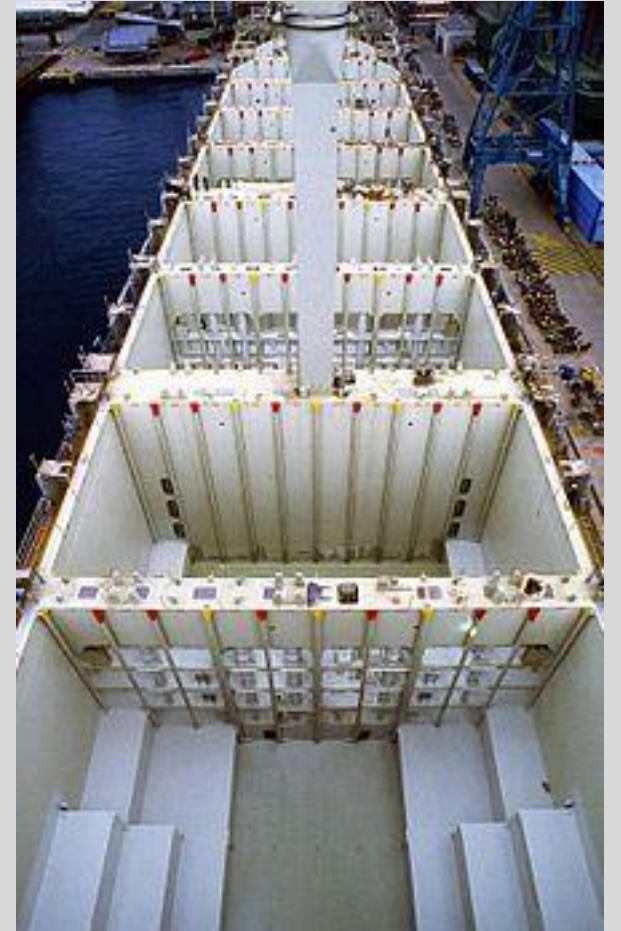
Services/Pods/Nodes in action



References: [#1](#), [#2](#)

Kubernetes Namespaces

- A [Kubernetes namespace](#) provides the scope for **Pods**, **Services**, and **Deployments** in the cluster. Users interacting with one namespace do not see the content in another namespace. Specify a namespace when deploying a resource.
- A Namespace specifies ..
 - resources (pods, services, replication controllers, etc.)
 - policies (who can or cannot perform actions in their community)
 - constraints (this community is allowed this much quota, etc.)
- Example Usage Scenarios:
 - Separate development resources from production resources.
 - Separate resources between different projects, teams, or customers.
 - Delegate authority to a set of resources to trusted users in those project team.
 - Limit the amount of resources each project can consume.
- low-level resources, such as **nodes** and **persistentVolumes**, are not in any namespace



Kubernetes - Pod Security Policy

- [A Pod Security Policy](#) controls security sensitive aspects of the pod specification. Defines a set of conditions that a pod must run with in order to be accepted into the system.

Running of privileged containers	user and group IDs of the container
Use of host namespaces (PID/IPC)	Restrict escalation to root privileges
Use of host networking & ports	Linux capabilities
Usage of volume types	The SELinux context of the container
Usage of the host filesystem	Allowed Proc Mount types
Allow specific FlexVolume drivers	AppArmor profile used by containers
Allocate an FSGroup for pod's volumes	seccomp profile used by containers
Require a read only root file system	The sysctl profile used by containers

Pod Security Policy / Context

A pod security context allows you to define runtime restrictions & settings on a per-pod basis. A namespace admin can define and enforce those security context-related policies using PSPs.

- To enable a Pod Security Policy you need to **create the policy**, then create a **cluster role** with permissions to use the policy. Create a **user / Pod Service** account and assign to the role.
- You must ensure that all users have access to a PSP. To do that sanely, you [grant all users access to the most restrictive PSP](#). But you could end up missing users added after PSPs are enabled, so grant the restrictive PSP to the **system:authenticated** group.
- PodSecurityPolicies are enforced by [enabling the admission controller](#). (intercepts requests to the Kubernetes API server). The admission controller acts on creation and modification of the pod and determines if it should be admitted based on the requested security context and the available Pod Security Policies.
- Kubernetes has a [default policy](#) defined, to apply if you wish.
- Be careful applying policies to existing running containers, they are typically created indirectly as part of a Deployment, ReplicaSet. References: [[Pod Security Policy](#), [Pod Security Context](#)]

Host Isolation Supported by Kubernetes

- PSP's include support the following Linux Kernel isolation options



Isolation Options on the OS Kernel

[Security Enhanced Linux \(SELinux\)](#): Uses labels to associate files, processes and ports. Format: user:role:type:level [Android, all Linux distributions]

[Linux Capabilities](#): Provide a subset of the available root privileges to a process. capabilities gives a binary root permissions for only a limited set of systems calls [[source](#)]

[AppArmor](#): Use program profiles to restrict the capabilities of individual programs. [Deb/Ubuntu/SUSE]

[Seccomp](#): Filter a process's system calls.

CIS Kubernetes Benchmark

- URL: <https://www.cisecurity.org/benchmark/Kubernetes/>
- Control Plane:
 - Apply file restrictions on Master Node
- Kubernetes API Server:
 - Don't allow anonymous auth / basic-auth / token auth
 - Authorization mode != AlwaysAllow
 - Enable RBAC authorization
 - Do not allow Admission controller AlwaysAdmin
 - Set AdmissionController plugin PodSecurityPolicy
 - Set SecurityContextDeny (if PSP's not being used)
- Worker Nodes
 - Ensure kubelet service and proxy kubeconfig file permissions are restricted and owned by root:root
 - Ensure kubelet anonymous-auth is false



Attack Vectors

App Compromise / Cluster Compromise

- Every container on the cluster is part of the cluster attack surface.
- The more applications deployed the larger the attack surface becomes
- Possible container compromise vectors.
 - DeSerialisation
 - File upload
 - Misconfiguration
 - Vulnerable package / dependency
 - Deployed an untrusted / tampered container
 - Server Side Include
 - SSH / RDP
- Kubernetes API Server Internet reachable / discovered by [Shodan](#)
- Important to Assume Breach and worry about containment



Docker Container Breakout



- Assume Breach of the App in a container. What then??
- Docker
 - Is the service running as root?
 - Are there folders mounted?
- Resources
 - [F-Secure](#), [root users](#), [Hacking and Hardening Kubernetes Clusters by Example](#)
- Using bind mounts to link an arbitrary files and folder to /etc.
- Run as non root
 - `docker run --user 1000 -v /:/home/notImportantDir/ innocent-docker-image`
- **APT remove <packagename>**
 - -> uninstall script running as root
 - --> uses a symlink in their config file used to delete a directory anywhere
 - ----> create a bind mount from the programX folder to /etc/

A **malicious user** with a shell in a container

By **default**, can **very possibly**

1. Exfiltrate source code, keys, tokens, and credentials
2. Elevate privileges inside Kubernetes to **access all workloads**
3. Gain **root access** to the underlying cluster nodes
4. Compromise other systems and data in the cloud account **outside the cluster**



K8S Attack Vectors

- **Service Account Token**

- When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace.
- A pod is associated with a service account, and a credential (token) for that service account is placed into the filesystem of each container in that pod: </var/run/secrets/kubernetes.io/serviceaccount/token>
 - If RBAC is not enabled .. every container has a **cluster admin token** mounted inside it.
 - If RBAC a service token is available for your **service account** used to deployed the service.
- This should not be a default. Applications do not typically need to interact with the Kubernetes API.
- In version **1.6+**, you can also [opt out of automounting](#) API credentials for a particular pod
- Deploy services using [multiple Service Accounts](#)

- Kubelet API needs auth enabled but previously had no authentication.

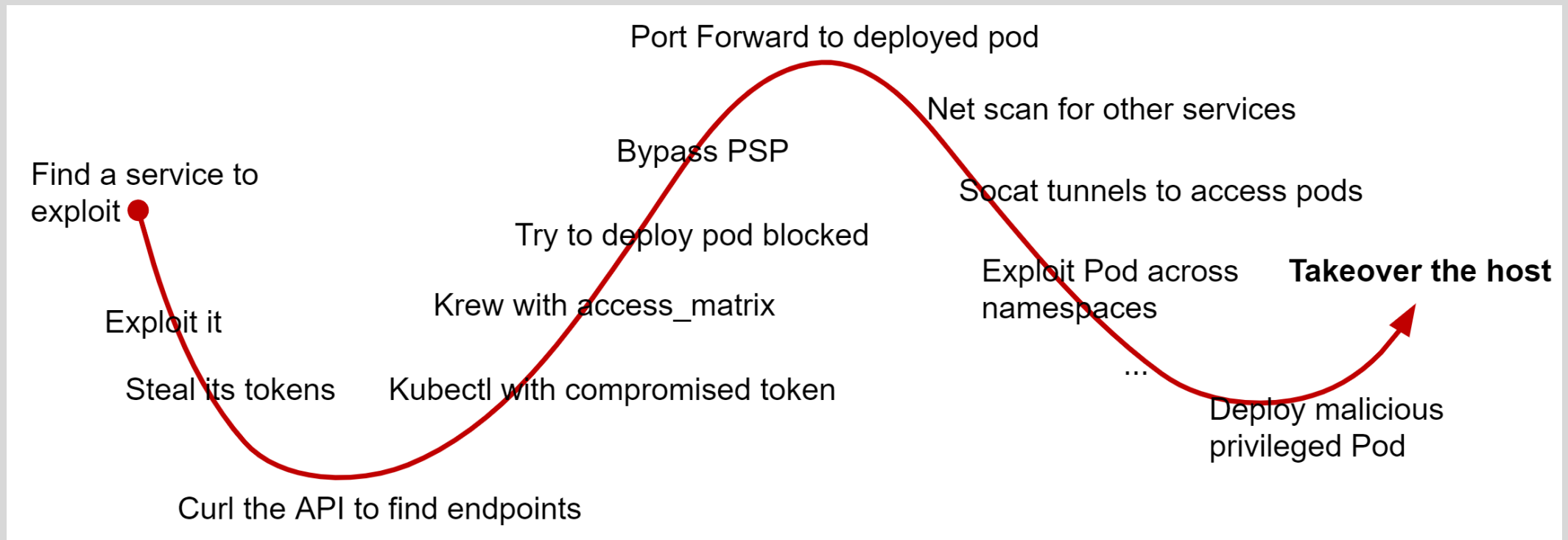
- Allows you to execute any command in any container on that node.

- ETCD

- Auth can be disabled. Has not always been configured with auth by default. Sensitive contents

- Steal Credentials from Kubernetes Secret store

NCC Attack Chain



NCC Talk: [Command and Kubectl](#)

Attack Chain (NCC)

Deep Dive into Real World Kubernetes Threats

Pod compromise	RCE into the Pod Steal Service Tokens Find the public IP of the cluster Setup Kubectl with the compromised token Determine what you can do in the cluster Try to deploy a Pod but get blocked
Namespace compromise	Bypass the PSP to deploy a Pod Port Forwarding into the Pod Finding other services in the cluster
Namespace tenant bypass	Compromise the other Pod Steal account token in new namespace Deploy a Privileged pod
Node compromise	Compromise the Node Deeper Compromise
Cluster compromise	Stealing kubelet config Create mirror pods Access the shell in the kube-system namespace

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

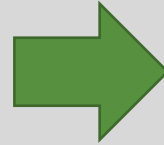
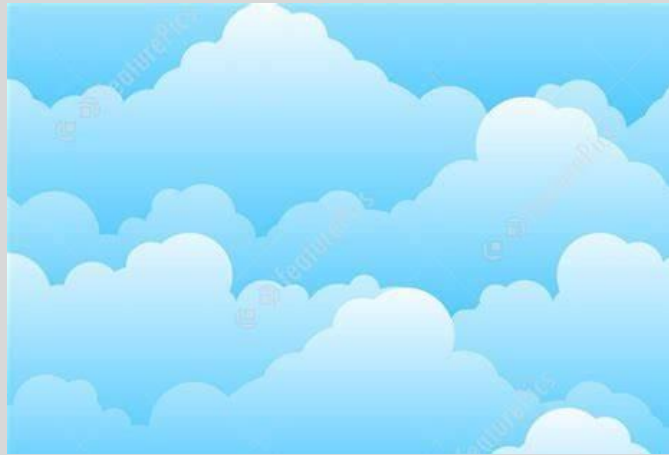
[Microsoft Kubernetes Threat Matrix](#)

1. "initial access to the computer" becomes "initial access to the cluster",
2. "malicious code on the computer" becomes "malicious activity on the containers",
3. "maintain access to the computer" becomes "maintain access to the cluster",
4. "gain higher privileges on the computer" becomes "gain higher privileges in the cluster".

Defences (NCC)

1. Don't allow privileged Pods
2. Don't allow a container to become root
3. Don't allow host mounts at all
4. Consider a network plugin or Network Policy for segmentation
5. Only use images and registries that you trust and don't rely on Dockerhub as a trusted source
6. Keep roles and role bindings as strict as possible
7. Don't automount Service Tokens into Pods if your services don't need to communicate to the API
8. Consider abstracting direct console access to the cluster away (ie Terraform, Spinnaker) so that none of your developers have cluster-admin permission.

Cloud Considerations



Clustering Considerations - AKS

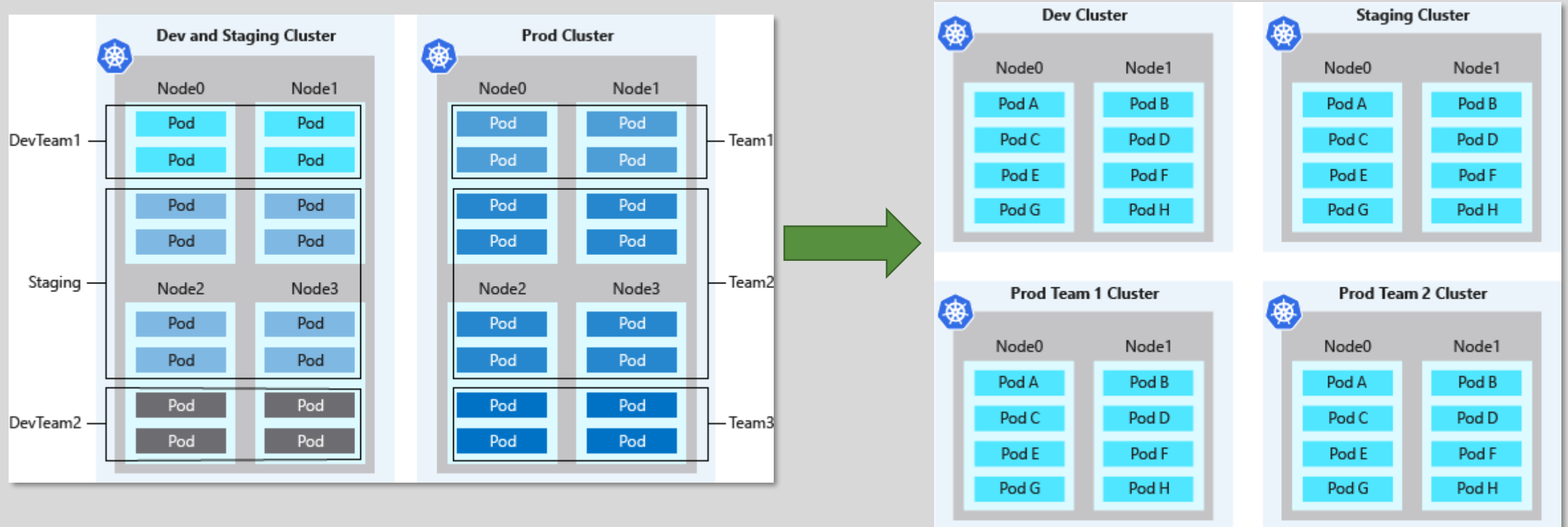
Goal of clustering is to maximise hardware

Security Principles

- Isolation
- Separation of duties
- Isolated identities
- Least Privilege
- Network Segmentation
- Benefits of individual clusters for applications
 - Can delete / decommission all app resources more easily
 - Cost of an application can be determined very accurately
 - Monitoring of events can be more easily attributed to an individual application
 - Good hygiene if all user accounts are user level of user level ...
 - Simplify access management
 - Simply DevOps. There is complexity in Kubernetes

AKS: Isolate by Team & Project

- [Cluster Isolation](#): Try to minimize the number of physical AKS clusters you deploy to isolate **teams** or **applications**.



AKS Considerations

- Provision **Private** Container Repository, **Private** Clusters
- Visibility of containers? Nodes should only allow connections from control plane on expected ports.
- Don't use the Kubernetes Credential Store – use [AAD POD Identity](#) plugin to access KeyVault
- Install the **Azure Policy Add-on** for AKS to allow the pod security settings to be defined in **Azure policies** (Linux nodes only). Involves the installation of [Gatekeeper](#) Admission Controller
- **Pod Security Policies** Supported by Azure: [Pod Security Policy for AKS](#)
 - No privileged containers, No running as root, Restrict **Linux capabilities** to the default set,
 - Disallow shared usage of host Namespaces, restrict defined volume types, Require **Seccomp Security profile**
 - **Readonly mounts, AppArmor Security Profile**
- **All policies default to an audit effect. Effects can be updated to deny at any time through Azure Policy.**
- Ensure Pod Security Policies are applied across all Pods on the cluster.

AKS

- AKS Pod Security Policies do **NOT Support**:
 - Defining custom **SELinux context** of a container
 - Restricting the **sysctl profile** used by containers
 - Defining default Proc Mount types
- **Namespace Exclusions**: new deployments to these namespaces to be **excluded from Azure policies**.
 - kube-system
 - gatekeeper-system
 - azure-arc
 - aks-periscope
- Limit Account Scope in Azure

Azure Resource Identity Type	Scope	Detail
System-Assigned Managed Identity	Limited to a Resource instance	Most desirable, created & deleted with the resource
User Assigned Managed Identity	Defined in a subscription	Use is restricted to within a subscription, needs to be deleted
Application Service Principal	Defined in a single Tenant	Least desirable, a tenant wide credential, created with an application registration

CICD Considerations

- Deploy frequently ..
- Ephemeral – the build pipeline even becoming containerised
- Output is containers, new deployment unit
- Scanning of dependencies. – [trivy](#)
- Deploy Kube-Hunter to prod
- Runtime scanning to detect drift from secure baseline, trigger redeployment

Closing Comments

- Containers
 - Origins and contents of containers needs continuous attention – build and runtime.
 - Minimise contents of a container, Try [distroless images](#).
 - Assume Breach of the container and think about the next lines of defence.
- Clusters
 - Deploy Private clusters, Private Container Repository
 - Don't overload a cluster with many different applications.
 - All containers are part of the cluster's attack surface.
 - Kubernetes is not secure by default.
 - Small Cluster: easier to enable & manage **RBAC**. Limit permissions.
 - Think of the Service Account Token's permissions!!
 - Easier to enable and manage Pod Security Policies – apply to all containers
 - Validate Policy against CIS Benchmarks (non privileged containers, read only disk etc)
- Cloud
 - Isolate different clusters into different subscriptions / cloud accounts
 - Monitor runtime environment for errors / abuse



References

Containers – [Docker for Beginners](#)

Security Guidance

- [Docker Security Homepage](#)
- [Security of Containers](#)
- [Kubernetes Security Homepage](#)
- CIS Benchmarks: [Docker](#), [Kubernetes](#), Azure, etc
- YouTube: [Kubernetes Security Best Practices](#) (Google)

Threat Awareness

- YouTube: [Hacking and Hardening Kubernetes Clusters by Example](#) (Brad Geesaman, Symantec)
- [Docker Mounting Abuse](#) (F-Secure)
- [Deep Dive into Kubernetes Threats](#) (Mark Manning, NCC)
- [Tools and Method for auditing Kubernetes RBAC Policies](#) (Mark Manning, NCC)
- [Walls within Walls](#) (Google) ([YouTube](#))





Options to Protect the Linux Kernel

SELinux

- Security Enhanced Linux – used on Android
- Using security policies, defines a set of rules that tell SELinux what can or can't be accessed
- SELinux works as a labeling system, which means that all of the files, processes, and ports in a system have an SELinux label associated with them. Labels are a logical way of grouping things together. The kernel manages the labels during boot.
- Labels are in the format **user:role:type:level** (level is optional). [\[ref\]](#)
- Kubernetes allows SELinux labels to be [assigned to a container](#) – configured in the Pod Security Context

```
...
securityContext:
  selinuxOptions:
    level: "s0:c123,c456"
```

Linux Capabilities

Provide a subset of the available root privileges to a process. capabilities gives a binary root permissions for only a limited set of systems calls

E.g. A web server wants to run on port 80. To listen on one of the lower ports (<1024), you need root permissions.

Instead of giving this daemon all root permissions, we can set a capability on the related binary, like CAP_NET_BIND_SERVICE. With this specific capability, it can open up port 80.

Kubernetes

- allows capabilities to be defined in PSP

```
My current capabilities are:
Current: =
Bounding set
=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,c
ap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_ne
t_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,
cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_
boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_lease,cap
_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_sy
slog,cap_wake_alarm,cap_block_suspend,37
Securebits: 00/0x0/1'b0
secure-noroot: no (unlocked)
secure-no-suid-fixup: no (unlocked)
secure-keep-caps: no (unlocked)
uid=1001(test)
gid=1001(test)
groups=1001(test)
```

AppArmor Security Profiles

- AppArmor is a [Linux kernel security module](#) that allows a sysadmin to restrict programs's capabilities with per-program profiles. AppArmor is offered in part as an alternative to [SELinux](#)
- AppArmor's security model is to bind access control attributes to programs rather than users.
- SELinux is based on applying labels to files, AppArmor works with file paths.
- Proponents of AppArmor claim that it is easier to learn than SELinux
- AppArmor includes a learning mode - violations are logged but not prevented.
- TCPDump [Example](#):
 - various access controls for files are present. From the profile we see 'r' (read), 'w' (write), 'm' (memory map as executable)
 - access controls for **capabilities** are present, access controls for **networking** are present
- Kubernetes docs have example involving [– SSH onto nodes to install](#) AppArmor profiles!!

```
#include <tunables/global>

/usr/sbin/tcpdump {
    #include <abstractions/base>
    #include <abstractions/namespace>
    #include <abstractions/user-tmp>

    capability net_raw,
    capability setuid,
    capability setgid,
    capability dac_override,
    network raw,
    network packet,

    # for -D
    capability sys_module,
    @{PROC}/bus/usb/ r,
    @{PROC}/bus/usb/** r,

    # for -F and -w
    audit deny @{HOME}/.* mrwkl,
    audit deny @{HOME}/.* / rw,
    audit deny @{HOME}/.* /** mrwkl,
    audit deny @{HOME}/bin/ rw,
    audit deny @{HOME}/bin/** mrwkl,
    @{HOME}/ r,
    @{HOME}/** rw,

    /usr/sbin/tcpdump r,
}
```

Seccomp Security Profiles

- **Secure Compute Mode** (a.k.a. seccomp) focuses on **limiting what system calls** your containers will be able to execute. has been a feature of the Linux kernel since version 2.6.12
 - defines what system calls should be allowed or blocked,
 - the container runtime will apply them at container start time so the kernel can enforce it.
 - Goal is to reduce the [kernel's attack surface](#).
- Supported by [Docker](#) (has a default system call [allowlist](#) published)
- [Kubernetes](#) lets you automatically apply seccomp profiles loaded onto a Node to your Pods and containers.
- A container can check if the host is running seccomp [\[ref\]](#):
 - **/ # grep Seccomp /proc/\$\$/status**
 - **Seccomp: 0**

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "name": "accept",
      "action": "SCMP_ACT_ALLOW",
      "args": []
    },
    {
      "name": "accept4",
      "action": "SCMP_ACT_ALLOW",
      "args": []
    },
    {
      "name": "access",
      "action": "SCMP_ACT_ALLOW",
      "args": []
    },
    {
      "name": "alarm",
      "action": "SCMP_ACT_ALLOW",
      "args": []
    },
    {
      "name": "bind",
      "action": "SCMP_ACT_ALLOW",
      "args": []
    },
    {
      "name": "brk",
      "action": "SCMP_ACT_ALLOW",
      "args": []
    }
  ]
}
```


Sysctl Security Profile

- Sysctl is an interface that allows you to make changes to a running Linux kernel. [/etc/sysctl.conf]
- Advanced security options of the TCP/IP stack and virtual memory to improve security & performance.
- Examples:
 - Limit network-transmitted configuration for IPv4
 - Limit network-transmitted configuration for IPv6
 - Turn on execshield protection
 - Prevent against the common 'syn flood attack'
 - Turn on source IP address verification
 - Logs several types of suspicious packets, such as spoofed packets, source-routed packets, and redirects.
 - Disable IP forwarding
 - Disable Send Packet Redirects
 - Disable ICMP Redirect Acceptance
 - Enable Bad Error Message Protection