Top Nine Kubernetes Settings You Should Check Right Now to Maximize Security

If you use Kubernetes, you know how much it can increase development velocity and reduce operational complexity. But did you know that Kubernetes settings can also help you secure your applications? Taking advantage of orchestrator security features is one of the highest-impact steps you can take to improve your security posture.

Unfortunately, your cluster might be less secure than it could be because:

- Your cluster was created when default configurations were less secure, and they're "frozen in time";
- You haven't opted in to new security features; or
- Your deployment strategy doesn't take advantage of the various types of security boundaries Kubernetes provides.

Check these nine configurations right now to ensure your Kubernetes cluster is configured for maximum security.

1. Upgrade to the Latest Version

New security features — and not just bug fixes — are added to every quarterly update, and to take advantage of them, we recommend you run the latest stable version. The very best thing to do is to run the latest release with its most recent patches, but if you can't do that, at least run the last one or two versions. Upgrades and support can become more difficult the farther behind you fall, so plan to upgrade at least once per quarter. Using a managed Kubernetes provider can make upgrades very easy.

Check your version now:

```
$ kubectl version --short
Client Version: v1.8.6
Server Version: v1.8.10-gke.0
```

2. Enable Role-Based Access Control (RBAC)

Control who can access the Kubernetes API and what permissions they have with Role-Based Access Control (RBAC). RBAC is usually enabled by default in Kubernetes 1.6 and beyond (later for some managed providers), but if you have upgraded since then and haven't changed your configuration, you'll want to double-check your settings. Because of the way Kubernetes authorization controllers are combined, you must both enable RBAC and disable legacy Attribute-Based Access Control (ABAC).

If you are running in Google Container Engine, you can check this setting using `gcloud`:

```
$ gcloud container clusters list –
format='table[box](name,legacyAbac.enabled)'
```

NAME	ENABLED
with-rbac	
with-abac	True

Once RBAC is being enforced, you still need to use it effectively. Cluster-wide permissions should generally be avoided in favor of namespace-specific permissions. Avoid giving anyone cluster admin privileges, even for debugging — it is much more secure to grant access only as needed on a case-by-case basis.

You can explore the cluster roles and roles using `kubectl get clusterrolebinding` or `kubectl get rolebinding --all-namespaces`. Quickly check who is granted the special "cluster-admin" role; in this example, it's just the "masters" group:

```
$ kubectl describe clusterrolebinding cluster-admin
Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:masters
```

If your application needs access to the Kubernetes API, create service accounts individually and give them the smallest set of permissions needed at each use site. This is better than granting overly broad permissions to the default account for a namespace.

Most applications don't need to access the API at all; `automountServiceAccountToken` can be set to "false" for these.

3. Use Namespaces to Establish Security Boundaries

Creating separate namespaces is an important first level of isolation between components. We find it's much easier to apply security controls such as Network Policies when different types of workloads are deployed in separate namespaces.

Is your team using namespaces effectively? Find out now by checking for any non-default namespaces:

```
kubectl get ns
NAME          STATUS    AGE
default       Active    16m
kube-public   Active    16m
kube-system   Active    16m
```

4. Separate Sensitive Workloads

To limit the potential impact of a compromise, it's best to run sensitive workloads on a dedicated set of machines. This approach reduces the risk of a sensitive application being accessed through a less-secure application that shares a container runtime or host. For example, a compromised node's kubelet credentials can usually access the contents of secrets only if they are mounted into pods scheduled on that node — if important secrets are scheduled onto many nodes throughout the cluster, an adversary will have more opportunities to steal them.

You can achieve this separation using node pools (in the cloud or on-premises) and Kubernetes namespaces, taints, tolerations, and other controls.

5. Secure Cloud Metadata Access

Sensitive metadata, such as kubelet admin credentials, can sometimes be stolen or misused to escalate privileges in a cluster. For example, a recent Shopify bug bounty disclosure detailed how a user was able to escalate privileges by confusing a microservice into leaking information from the cloud provider's metadata service. GKE's metadata concealment feature changes the cluster deployment mechanism to avoid this exposure, and we recommend using it until it is replaced with a permanent solution. Similar countermeasures may be needed in other environments.

6. Create and Define Cluster Network Policies

Network Policies allow you to control network access into and out of your containerized applications. To use them, you'll need to make sure that you have a networking provider that supports this resource; with some managed Kubernetes providers such as Google Kubernetes Engine (GKE), you'll need to opt in. (Enabling network policies in GKE will require a brief rolling upgrade if your cluster already exists.) Once that's in place, start with some basic default network policies, such as blocking traffic from other namespaces by default.

If you are running in Google Container Engine, you can check whether your clusters are running with policy support enabled:

```
$  gcloud container clusters list --
format='table[box](name,addonsConfig.networkPolicyConfig)'
┌──────────────┬──────────────────────────┐
│     NAME     │ NETWORK_POLICY_CONFIG    │
├──────────────┼──────────────────────────┤
│ without-policy │ {u'disabled': True}    │
│ with-policy    │ {}                     │
└──────────────┴──────────────────────────┘
```

In other environments, check that you're using one of the network providers that support policies. (https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/)

Once you know policies are supported, check to see if you have any:

```
$ kubectl get networkpolicies --all-namespaces
No resources found.
```

7. Run a Cluster-wide Pod Security Policy

A Pod Security Policy sets defaults for how workloads are allowed to run in your cluster. Consider defining a policy and enabling the Pod Security Policy admission controller — instructions vary depending on your cloud provider or deployment model. As a start, you could require that deployments drop the NET_RAW capability to defeat certain classes of network spoofing attacks.

8. Harden Node Security

- **Ensure the host is secure and configured correctly**. One way to do so is to check your configuration against CIS Benchmarks; many products feature an autochecker that will assess conformance with these standards automatically.
- **Control network access to sensitive ports.** Make sure that your network blocks access to ports used by kubelet, including 10250 and 10255. Consider limiting access to the Kubernetes API server except from trusted networks. Malicious users have abused access to these ports to run cryptocurrency miners in clusters that are not configured to require authentication and authorization on the kubelet API server.
- **Minimize administrative access to Kubernetes nodes**. Access to the nodes in your cluster should generally be restricted — debugging and other tasks can usually be handled without direct access to the node.

9. Turn on Audit Logging

Make sure you have audit logs enabled and are monitoring them for anomalous or unwanted API calls, especially any authorization failures — these log entries will have a status message “Forbidden.” Authorization failures could mean that an attacker is trying to abuse stolen credentials. Managed Kubernetes providers, including GKE, provide access to this data in their cloud console and may allow you to set up alerts on authorization failures.

Next Steps

Follow these StackRox recommendations for a more secure Kubernetes cluster. Remember, even after you follow these tips to configure your Kubernetes cluster securely, you will still need to build security into other aspects of your container configurations and their runtime operations. As you improve the security of your tech stack, look for tools that provide a central point of governance for your container deployments and deliver continuous monitoring and protection for your containers and cloud-native applications.