SHADOW SOFT

# Kubernetes: A CISO user guide using a cloud to code framework

Kubernetes intensive automation requires a new framework to secure disparate application, database, network, and storage components.

By Derrick Sutherland, Nick Marcarelli, and Kurt Baumberger

February 2023

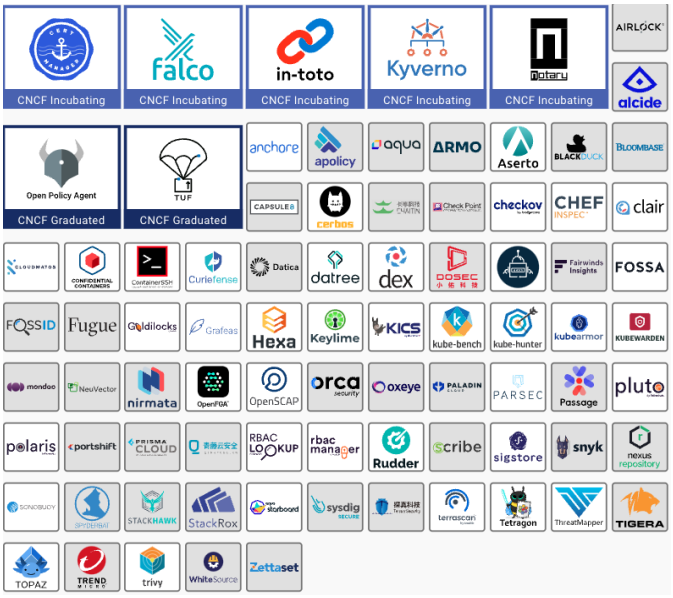
## Here's the challenge...

Kubernetes can seem like a journey into the unknown. For most organizations, Kubernetes requires taking paths not yet charted. This involves risk. But it doesn't need to.

Kubernetes is simply a Software Defined Datacenter for Containerized workloads. By providing a series of tightly coupled technology solutions, Kubernetes enables workloads to dynamically scale in a standardized way across various linux systems.

Fortunately, there is a clear path forward to securing your Kubernetes environment. The first step involves understanding core Kubernetes functionality. Next, you'll need a framework that we call the "4 C's" to think through your security considerations. Then, you'll need to build a comprehensive series of maps in a Kubernetes Health Assessment which will address eight (8) specific security issues.

The good news is there are plenty of tools available to lock down your infrastructure. The better news is that it only takes a few weeks of concentrated effort to build a scalable secure environment. The best news is there is a time-tested, proven approach to simplify the complexity of Kubernetes.




Like with most endeavors, preparation is the key to success. Don't skip this step on your Kubernetes journey. Any successful expedition requires good preparation in order to avert disaster down the road.

So let's get started...

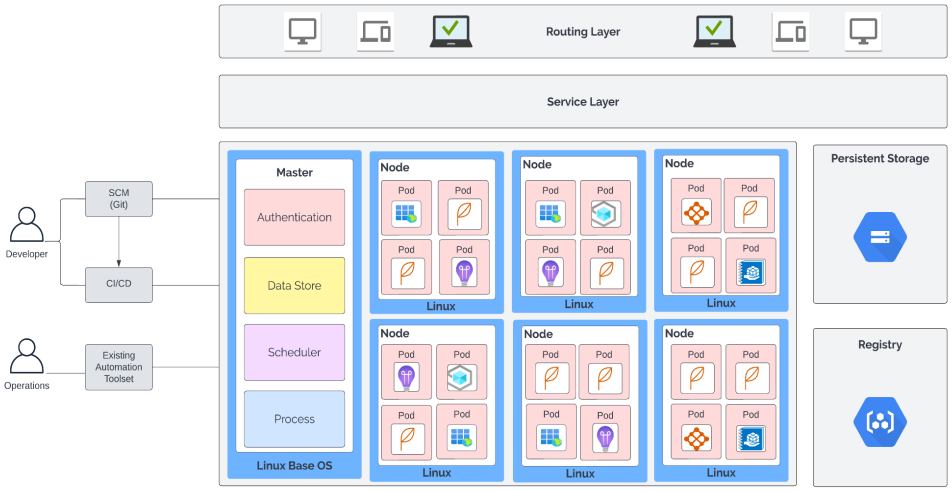
# Kubernetes 101

The secret to Kubernetes speed, efficiency, and compute power lies in standardizing baseline functionality, system constructs, and API definitions to create a consistent experience regardless of your distribution selection.

Baseline functionality includes:

- Container Workload Scheduling & Recovery
- Configuration State Management
- Remote Management API Interface
- User Authentication/Authorization
- Security Policy Management
- Software Defined Networking
- Standardized Storage Support
- Universal Plug-N-Play Support for Various Infrastructure Providers

Here is a visual representation of Kubernetes components to help you understand how each component is related and integrated with another.




To ensure workloads (and the environment itself) are designed in a resilient and secure manner, it is important to understand the basic mechanics of how Kubernetes allocates & schedules deployments. This is particularly important because there are many configurable parameters that impact how and where workloads are scheduled and deployed.

Unlike traditional virtual machine workloads which are long-lived and often point-solution specific, Kubernetes workloads have distinct characteristics. These characteristics will shape your security approach and posture.

- ☑ Shared host OS resources (i.e., Memory, CPU, Filesystem Resources)
- ☑ Shared Cluster resources (i.e., Storage, IP Allocation, Memory, CPU)
- ☑ Frequent workload horizontal scaling events
- ☑ Frequent node horizontal scaling events
- ☑ Frequent changes to IP addresses
- ☑ Frequent process reload / restart events
- ☑ Ephemeral by default (including system logging and local storage)

To illustrate the dynamic nature of Kubernetes, consider what happens when a deployment health check fails. First, the current deployment will be terminated. Then, a new deployment instance will be created. Next, the new deployment will go through the same scheduling process as if it never previously ran.

The new instance is usually deployed on a new node. It will certainly be provisioned with a new IP address. The data (if any) generated from the original deployment instance is lost. This same behavior automatically occurs any time a node goes offline due to a restart, failure, or update event.

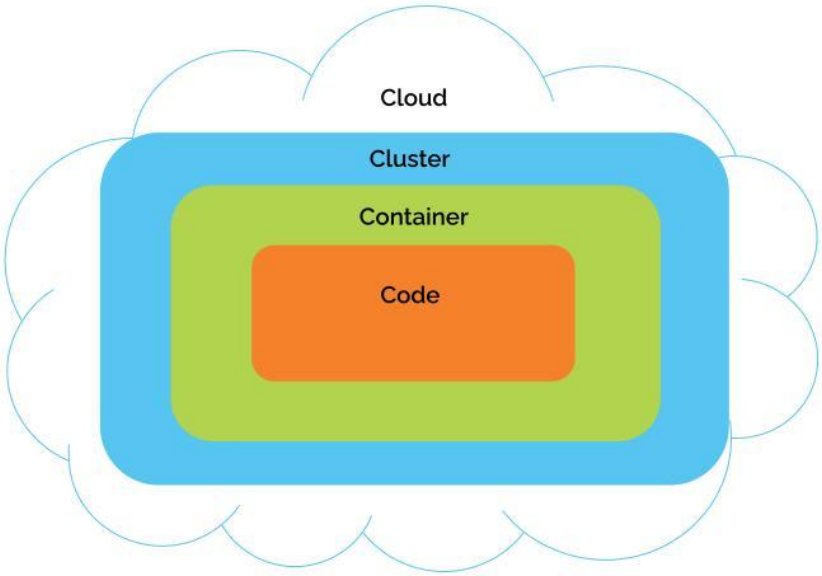
## 4C's Framework

As you can imagine, the security implications of an environment that self-heals, auto-scales, auto-replicates, and auto-deploys based on the resources available at the time requires a new way of thinking. Most security policies never contemplated ephemeral data or decentralized computing on the edge and yet these are the requirements that security must meet today.


To help migrate to a secure Kubernetes environment, a framework is needed. This framework can be used to gain consensus on what needs attention and provides a rationale for funding. Since most Kubernetes environments leverage the scalability and efficiency of the cloud, a "4 C's" framework makes sense for most organizations to use.

Simply put, you can think about Kubernetes security in four layers: ***Cloud, Clusters, Containers, and Code.***

Each layer of the Cloud Native security model builds upon the next outermost layer. The code layer benefits from strong base (Cloud, Cluster, Container) security layers. You cannot safeguard against poor security standards in the base layers by addressing security at the code level.




## *Cloud*

The Cloud (or co-located servers, or the corporate datacenter) is the trusted computing base of a Kubernetes Cluster. If the Cloud layer is vulnerable (or configured in a vulnerable way), then there is no guarantee that the components built on top of this base are secure. Each cloud provider makes recommendations for running workloads securely in their environment.

While each cloud provider has unique security provisions, here are some general guidelines for securing your infrastructure:

| **Security Concern** | **Recommendation** |
|---|---|
| Network access to API Server (Control Plane) | All access to the Kubernetes control plane is not allowed publicly on the internet and is controlled by network access control lists restricted to the set of IP addresses needed to administer the Cluster. |
| Network access to Nodes | Configure nodes to *only* accept connections from the control plane on the specified ports, and accept connections for services in Kubernetes of type NodePort and LoadBalancer. Do not expose these nodes on the public internet.. |

| | |
|---|---|
| Kubernetes access to Cloud Provider API | Each cloud provider needs to grant a different set of permissions to the Kubernetes control plane and nodes. It is best to provide the Cluster with cloud provider access that follows the principle of least privilege for the resources it needs to administer. |
| Access to etcd | Access to etcd (the datastore of Kubernetes) should be limited to the control plane only. Depending on your configuration, you should attempt to use etcd over TLS. |
| etcd Encryption | Wherever possible it's a good practice to encrypt all storage at rest, and since etcd holds the state of the entire Cluster (including Secrets), its disk should especially be encrypted at rest. |

## *Cluster*

There are two areas of concern for securing Kubernetes Clusters:

- Securing the Cluster components that are configurable
- Securing the applications which run in the Cluster

Depending on the attack surface of your application, you may want to focus on specific aspects of security. For example, if you are running a service (Service A) that is critical in a chain of resources and a separate workload (Service B) which is vulnerable to a resource exhaustion attack, then the risk of compromising Service A is high if you don't limit the resources of Service B.

Here are the seven Cluster security concerns that you must fully address:

1. RBAC Authorization (Access to the Kubernetes API)
2. Authentication
3. Application Secrets Management (etcd encryption and at rest)
4. Ensuring pods meet defined Pod Security Standards
5. Quality of Service and Cluster Resource Management
6. Network Policies
7. TLS for Kubernetes Ingress

Unfortunately, Clusters often break. The optimal way to address these seven Cluster concerns requires a comprehensive review of your existing environment, policies, and governance procedures. Only then can you ensure your Kubernetes architecture will scale securely and your organization is equipped with the tools they need to rapidly spin up applications at the speed of business.

## Containers

In simple terms, Containers decouple applications from underlying host infrastructure and this standardization means you get the same behavior wherever you run it. This is why your Kubernetes environment self-heals, auto-scales, auto-replicates, and auto-deploys.

Container images are ready-to-run software packages, containing everything needed to run an application: the Code and any runtime it requires, application and system libraries, and default values for any essential settings.

This self-sufficiency allows Containers to be stateless and immutable; you should not change the Code of a Container that is already running. If you have a containerized application and want to make changes, the correct process is to build a new image that includes the change, then recreate the Container to start from the updated image.

To optimize security, you should follow these recommendations:

| **Security Concern** | **Recommendation** |
|---|---|
| Container Vulnerability Scanning and OS Dependency Security | As part of a Container image build step, you should scan your Containers for known vulnerabilities. |
| Image Signing and Enforcement | Sign Container images to maintain a system of trust for the content of your Containers. |
| Disallow privileged users | When constructing Containers, consult your documentation for how to create users inside of the Containers that have the least level of operating system privilege necessary in order to carry out the goal of the Container. |
| Use Container runtime with stronger isolation | Select Container runtime classes that provide stronger isolation. |

## Code

Finally, we reach the area where you have the most control: Application Code. It is one of the primary attack surfaces and you probably have several tools, policies, and procedures in place. You will want to keep many of these existing security guidelines and only introduce a few new best practices that Kubernetes demands.

Here are some specific recommendations to protect your application code:

| Security Concern | Recommendation |
|---|---|
| Access over TLS only | If your code communicates by TCP, perform a TLS handshake with the client ahead of time. Encrypt everything in transit and network traffic between services using mutual TLS authentication (mTLS) which performs a two sided verification of communication between two certificate holding services. |
| Limiting port ranges of communication | Wherever possible, only expose the ports on your service that are absolutely essential for communication or metric gathering. |
| 3rd Party Dependency Security | Routinely scan your application's third party libraries for known security vulnerabilities. Each programming language has a tool to perform this check automatically. |
| Static Code Analysis | Most languages provide a way for a snippet of code to be analyzed for unsafe coding practices. Use automated tooling that can scan Codebases for common security errors. |
| Dynamic probing attacks | Use automated tools that you can run against your service to try some of the well known service attacks. These include SQL injection, CSRF, and XSS. |

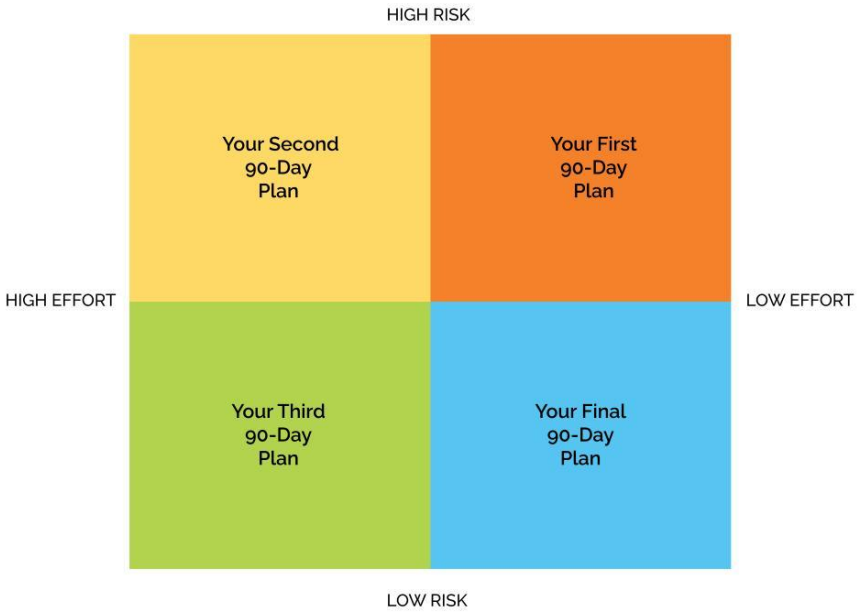
## Kubernetes Health Assessment

Unlike other platforms, Kubernetes provides its users the full functionality of a Software Defined Datacenter with high levels of automation. The best way to address these risks is to conduct a comprehensive Health Assessment on eight dimensions:

1. Architecture
2. Applications
3. Scalability
4. Storage & Backup
5. CI/CD
6. Disaster Recovery
7. Monitoring & Health Checks
8. Security

A good Health Assessment will leverage the power of maps. Maps will help you review what's happening now and decide what you need to do differently. They can also shape your path around, over, or through the security hazards identified in your 4 C's Framework: Cloud, Cluster, Container, Code.

More importantly, the maps will help you plot on a Risk x Effort Scorecard how to systematically strip out risk in a series of 90-Day Plans that produce a 12-Month Roadmap.




## What's Next?

A Health Assessment will take you and your team approximately two weeks to complete. To download a step-by-step Health Assessment Guide, go to www.Shadow-Soft.com. To take a free course on Kubernetes 101 or to learn how to conduct your own Kubernetes Health Assessment with Enterprise Architect office hours at no charge, go to www.academy.Shadow-Soft.com.

For other inquiries or support, please feel free to contact Nick Marcarelli, VP of Engineering, nick@shadow-soft.com. Nick can help you think through your specific issues and use case.

We're here to Make Optimal Possible.©