Overview

OpenShift v3 is a layered system designed to expose underlying Docker-formatted container image and Kubernetes concepts as accurately as possible, with a focus on easy composition of applications by a developer. For example, install Ruby, push code, and add MySQL.
Unlike OpenShift v2, more flexibility of configuration is exposed after creation in all aspects of the model. The concept of an application as a separate object is removed in favor of more flexible composition of "services", allowing two web containers to reuse a database or expose a database directly to the edge of the network.

What Are the Layers?

The Docker service provides the abstraction for packaging and creating Linux-based, lightweight container images. Kubernetes provides the cluster management and orchestrates containers on multiple hosts.

OpenShift Container Platform adds:

- Source code management, builds, and deployments for developers
- Managing and promoting images at scale as they flow through your system
- Application management at scale
- Team and user tracking for organizing a large developer organization

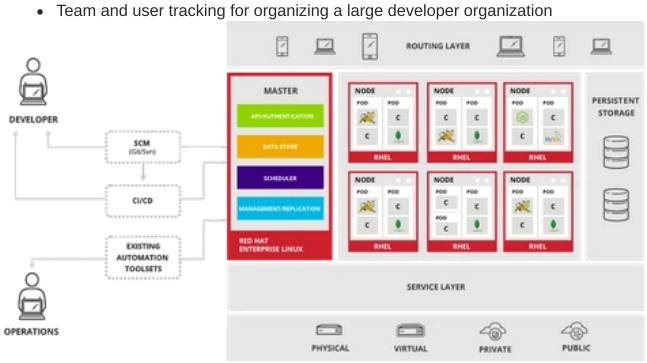

Figure 1. OpenShift Container Platform Architecture Overview

What Is the OpenShift Container Platform Architecture?

OpenShift Container Platform has a microservices-based architecture of smaller, decoupled units that work together. It runs on top of a Kubernetes cluster, with data about the objects stored in etcd, a reliable clustered key-value store. Those services are broken down by function:

- REST APIs, which expose each of the core objects.
- Controllers, which read those APIs, apply changes to other objects, and report status or write back to the object.

Users make calls to the REST API to change the state of the system. Controllers use the REST API to read the user's desired state, and then try to bring the other parts of the system into sync. For example, when a user requests a build they create a "build" object. The build controller sees that a new build has been created, and runs a process on the cluster to perform that build. When the build completes, the controller updates the build object via the REST API and the user sees that their build is complete.
The controller pattern means that much of the functionality in OpenShift Container Platform is extensible. The way that builds are run and launched can be customized independently of how images are managed, or how deployments happen. The controllers are performing the "business logic" of the system, taking user actions and transforming them into reality. By customizing those controllers or replacing them with your own logic, different behaviors can be implemented. From a system administration perspective, this also means the API can be used to script common administrative actions on a repeating schedule. Those scripts are also controllers that watch for changes and take action. OpenShift Container Platform makes the ability to customize the cluster in this way a first-class behavior.
To make this possible, controllers leverage a reliable stream of changes to the system to sync their view of the system with what users are doing. This event stream pushes changes from etcd to the REST API and then to the controllers as soon as changes occur, so changes can ripple out through the system very quickly and efficiently. However, since failures can occur at any time, the controllers must also be able to get the latest state of the system at startup, and confirm that everything is in the right state. This resynchronization is important, because it means that even if something goes wrong, then the operator can restart the affected components, and the system double checks everything before continuing. The system should eventually converge to the user's intent, since the controllers can always bring the system into sync.

How Is OpenShift Container Platform Secured?

The OpenShift Container Platform and Kubernetes APIs authenticate users who present credentials, and then authorize them based on their role. Both developers and administrators can be authenticated via a number of means, primarily OAuth tokensand SSL certificate authorization.
Developers (clients of the system) typically make REST API calls from a client program like `oc` or to the web console via their browser, and use OAuth bearer tokens for most communications. Infrastructure components (like nodes) use client certificates generated by the system that contain their identities. Infrastructure components that run in containers use a token associated with their service account to connect to the API.
Authorization is handled in the OpenShift Container Platform policy engine, which defines actions like "create pod" or "list services" and groups them into roles in a policy document. Roles are bound to users or groups by the user or group identifier. When a user or service account attempts an action, the policy engine checks for one or more of the roles assigned to the user (e.g., cluster administrator or administrator of the current project) before allowing it to continue.
Since every container that runs on the cluster is associated with a service account, it is also possible to associate secrets to those service accounts and have them automatically delivered into the container. This enables the infrastructure to manage secrets for pulling and pushing images, builds, and the deployment components, and also allows application code to easily leverage those secrets.

Core Concepts

The following topics provide high-level, architectural information on core concepts and objects you will encounter when using OpenShift Container Platform. Many of these objects come from Kubernetes, which is extended by OpenShift Container Platform to provide a more feature-rich development lifecycle platform.

- Containers and images are the building blocks for deploying your applications.
- Pods and services allow for containers to communicate with each other and proxy connections.
- Projects and users provide the space and means for communities to organize and manage their content together.
- Builds and image streams allow you to build working images and react to new images.
- Deployments add expanded support for the software development and deployment lifecycle.
- Routes announce your service to the world.
- Templates allow for many objects to be created at once based on customized parameters.