OpenShift is offered in three forms:

- **OpenShift Online** -- a cloud based, hosting service for application developers
- **OpenShift Enterprise** -- a PaaS platform designed to run within an organization's data center
- **OpenShift Origin** -- the open source application hosting platform underlying OpenShift Online and OpenShift Enterprise.

The focus of this article is OpenShift Online.

OpenShift, like other PaaS platforms, is designed for developers. Many typical systems administration tasks are automated so developers can spend more time on code and less time on configuring operating systems and installing libraries and packages. The most important of the automated administration tasks are **virtual server provisioning, configuration, and scaling**

Developers work at the level of applications and components. In OpenShift, an application is a combination of code, configurations and application components called cartridges. **Cartridges are high level services**, such as Web servers, databases, as well as logging and monitoring tools. Cartridges are logically isolated from one another (using a combination of Linux namespacing and Security Enhanced Linux (SELinux) features) so **multiple cartridges can run on the same server**. OpenShift uses the terms "**gears" and "nodes"** to refer to the abstract structures that isolate components. In OpenShift parlance, **nodes hold gears that contain one or more cartridges**. A broker manages provisioning and application management processes and communications with nodes over a message bus.

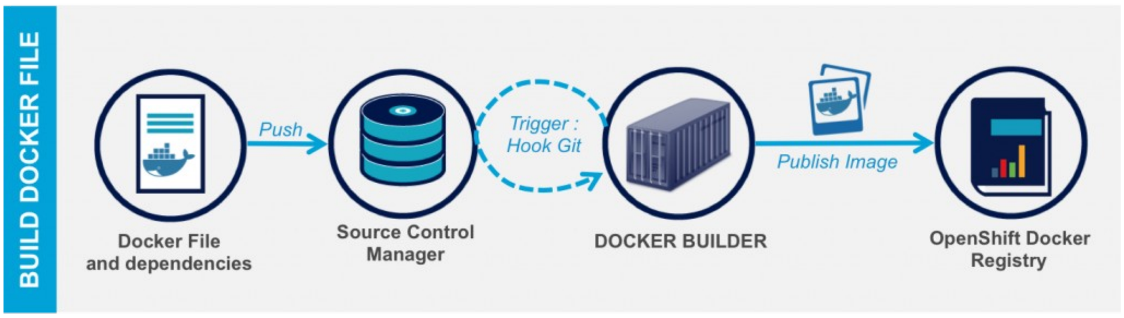
The genesis of the third version

Since July 2014, OpenShift has been working on an ambitious **refactoring project** of its technical architecture to integrate – the now inevitable – **Docker** and **Kubernetes**.
Launching this project a year ago was a notably **bold and risky strategy** for OpenShift. Indeed, while the competition with Cloud Foundry was at its peak, OpenShift chose to launch an **important reengineering project**with side effect of freezing development of new features and compromising compatibility with previous versions. We now believe they made the right choice.
To date, the Origin project federates 86 contributors on Github (of which the 10 most active work at **Red Hat**). This quite active community made about 16 iterations in 12 months and has just published its already very promising **first release**.
Although the solution is mainly driven by **Red Hat**, it heavily depends on **Kubernetes** (by **Google**). This raises the complex question of **governance** which depends on good collaboration and visions alignment of the two companies. What will happen with future features developed by Google: will they focus on Kubernetes or OpenShift? Obviously, the question is not yet settled.
However, if the support of Google is confirmed, a **two-headed governance** of these giants can only be beneficial to OpenShift. It will confirm its **competitive advantage over Docker Enterprise** (Machine, Compose and Swarm).

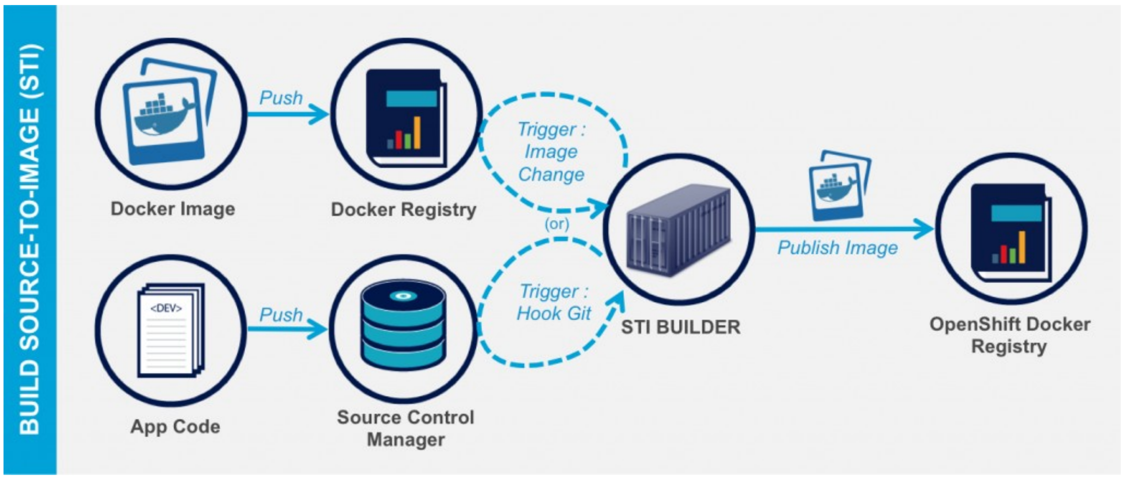
Automatically build and deploy applications: how it works

OpenShift v3 offers **3 native ways to build automatically** an application :

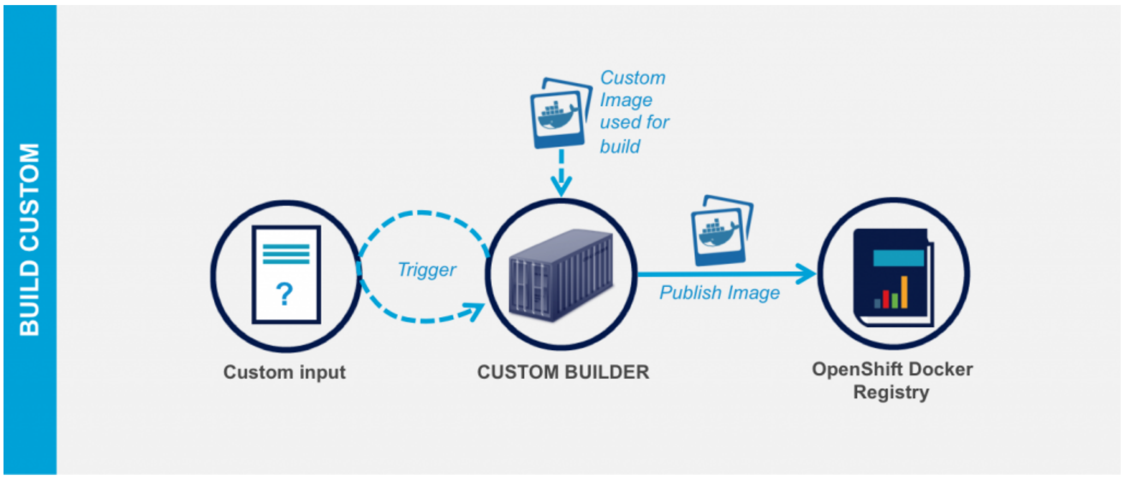
- **The Docker-File mode**: automatically build a Docker container by providing OpenShift with the URI to a source code manager pointing at a Docker-File and its dependencies.



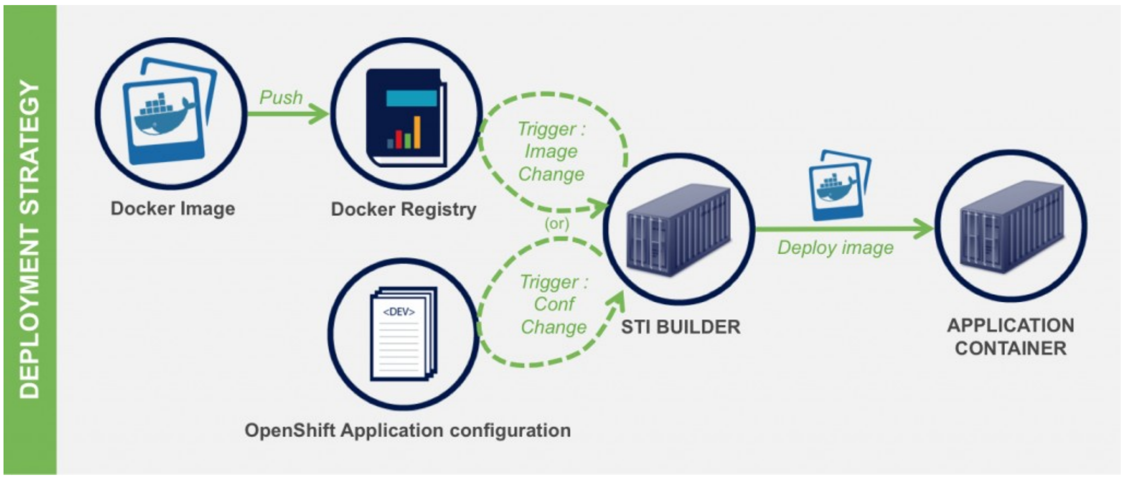
•**The Source-To-Image mode (STI)**: allows to automatically build an application by committing the application source code in OpenShift (like **buildpacks** in **Heroku**).



•**The Custom Build Mode**: allows to provide one's own application building logic by providing an OpenShift Docker image designed for this purpose.



OpenShift 3 also allows to define an **automated deployment strategy** of an application when a new **image version** is published in the registry or when the **configuration** of the application is updated.



To complete these **build and deployment features**, OpenShift 3 provides the ability to define its own **application blueprints** as template files described in Json or Yaml format. These blueprints describe both the **topology** of the application architecture and the **containers deployment policy**. The diagram below illustrates the assembly of the different components of a template for a 3-tier application in OpenShift.

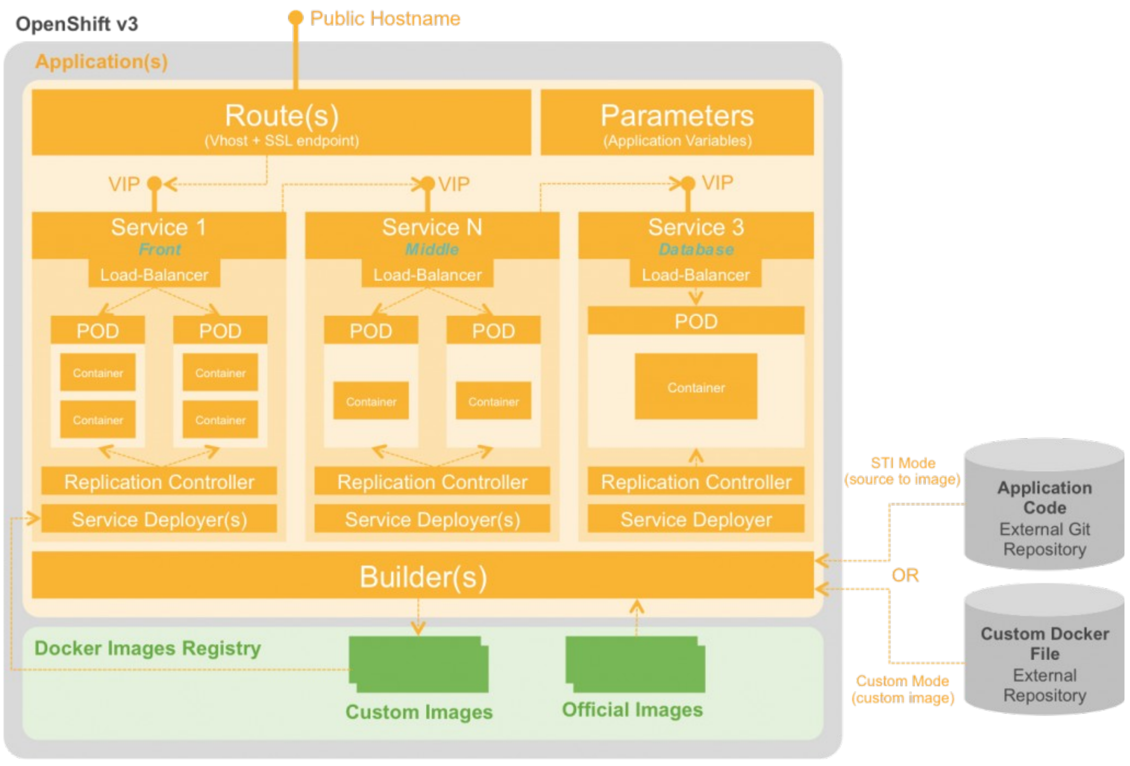


Diagram : example of a 3 tier application architecture hosted in OpenShift v3

The components assembled in a "template" are partially inherited from **Kubernetes concepts**. The main objects to remember are:

- A "**POD**" is a Docker container runtime environment local to a server (we will deploy two types of containers on a single POD if it is necessary to share local resources)
- A "**Service**" is an entry point (VIP) abstracting a load-balanced access to a group of identical containers. In principle, at least one Service per architecture tier is deployed
- A "**Service Deployer**" or "**Deployment Config**" is an object that describes a deployment policy of a container based on triggers (eg redeploy when a new version of an image is available in the registry

Docker)

- A “**Replication Controller”** is a technical component in charge of the resilience of POD
- A “**Route”** exposes an entry point (DNS hostname or VIP) outgoing of an application

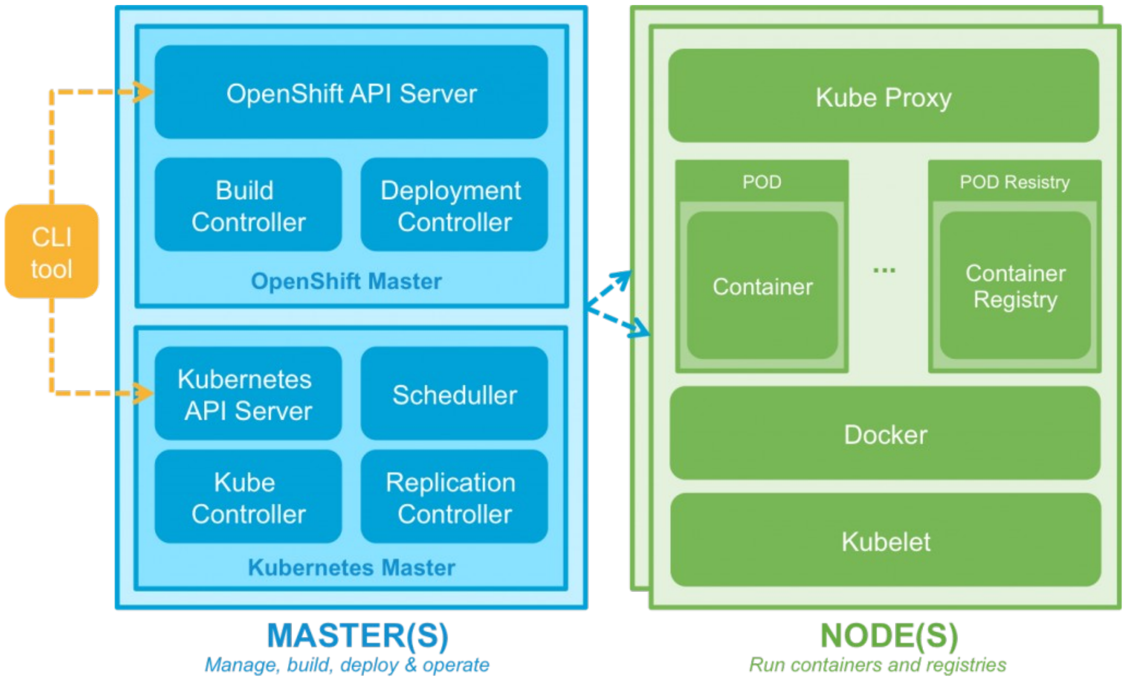
Through its **multiple deployment mechanisms** and the ability to set its **own “blueprints”** OpenShift v3 **comply**to the most **complex application** architectures.

architecture under the hood

A deployment of an OpenShift 3 infrastructure can be done in **standalone** mode (using the docker image “openshift/origin”), or in a **distributed** way using the **ANSIBLE** playbooks provided by OpenShift. In the latter case, two kinds of server **roles** are used: “Master” and “Node”.
The “Master” nodes purpose is to:

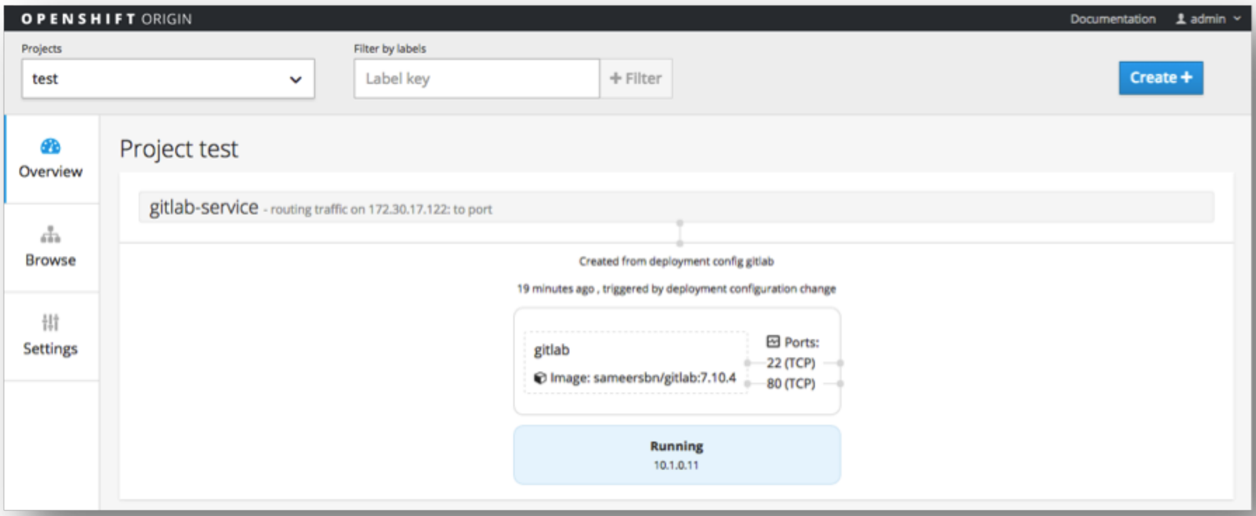
- process administration API requests coming from the CLI or the Web Portal
- build images and deploy containers
- ensure PODS resiliency through replication

“**Masters**” rely on a distributed directory based on etcd to provide **configuration sharing** and **services discovery**.
“**Nodes**” host the PODS and run **containers (applications** and/or ***Registry***).



Deployment Architecture of an Openshift v3 infrastructure

The architecture is at the same time **distributed, scalable and resilient**. However, automatic scalability is not yet supported by the platform itself: **provisioning and capacity planning of the underlying servers** currently have to be done **manually**.
The platform can be accessed and managed through its **REST API**, its **CLI** or its **Web portal** (but currently only in read-only mode for the latter).


OPENSHIFT ORIGIN
Documentation
admin
Projects
test
Filter by labels
Label key
+ Filter
Create +
Overview
Browse
Settings
Project test
gitlab-service - routing traffic on 172.30.17.122: to port
Created from deployment config gitlab
19 minutes ago , triggered by deployment configuration change
gitlab
Image: sameersbn/gitlab:7.10.4
Ports:
22 (TCP)
80 (TCP)
Running
10.1.0.11

OpenShift 3 offers an interesting bridge between the "*Plateform-as-a-Service*" world and the "*Container-as-a-Service*" (CaaS) world. Red Hat came up here with a **bold solution** and a **state-of-the-art architecture**. We strongly appreciate the "*blueprints" specification format for architecture definition and deployment orchestration.*
In the beta 3 version, OpenShift platform didn't put a strong focus on **platform operability**. It shouldn't be used in **production** yet. However, *user stories* on that matter can be found in the roadmap:

- ***log aggregation** with* ElasticSearch–Logstash–Kibana *or* Fluentd
- ***monitoring** with Heapster,*
- ***metrics collection**,*
- *ease of cluster **deployment**.*

OpenShift 3 is a new job that will **require new skills** in order to ask the appropriate questions: how to organize the containers? Should routes or services be used? How to handle data (persistence, replication, backup)? How to manage multi-tenancy? How to integrate Development and Deployment Software Factory?
To sum up, OpenShift is a **very promising private PaaS** solution. It allows **to reduce the time to market** by automating **application building and deployment processes** from the very beginning of the project. It **supports** the most **complex web architectures**, even if the topics of **data management** and external services integration are not fully addressed currently.
We believe Openshift 3 has everything in hand to **become a reference** in the area of **Docker based private PaaS**.