Published in Dev Genius

Tony Follow
Jan 20 · 5 min read · ✦ · Listen

Save

K8s — Robusta, K8s Troubleshooting Platform

Robusta, the open source K8s troubleshooting platform intro

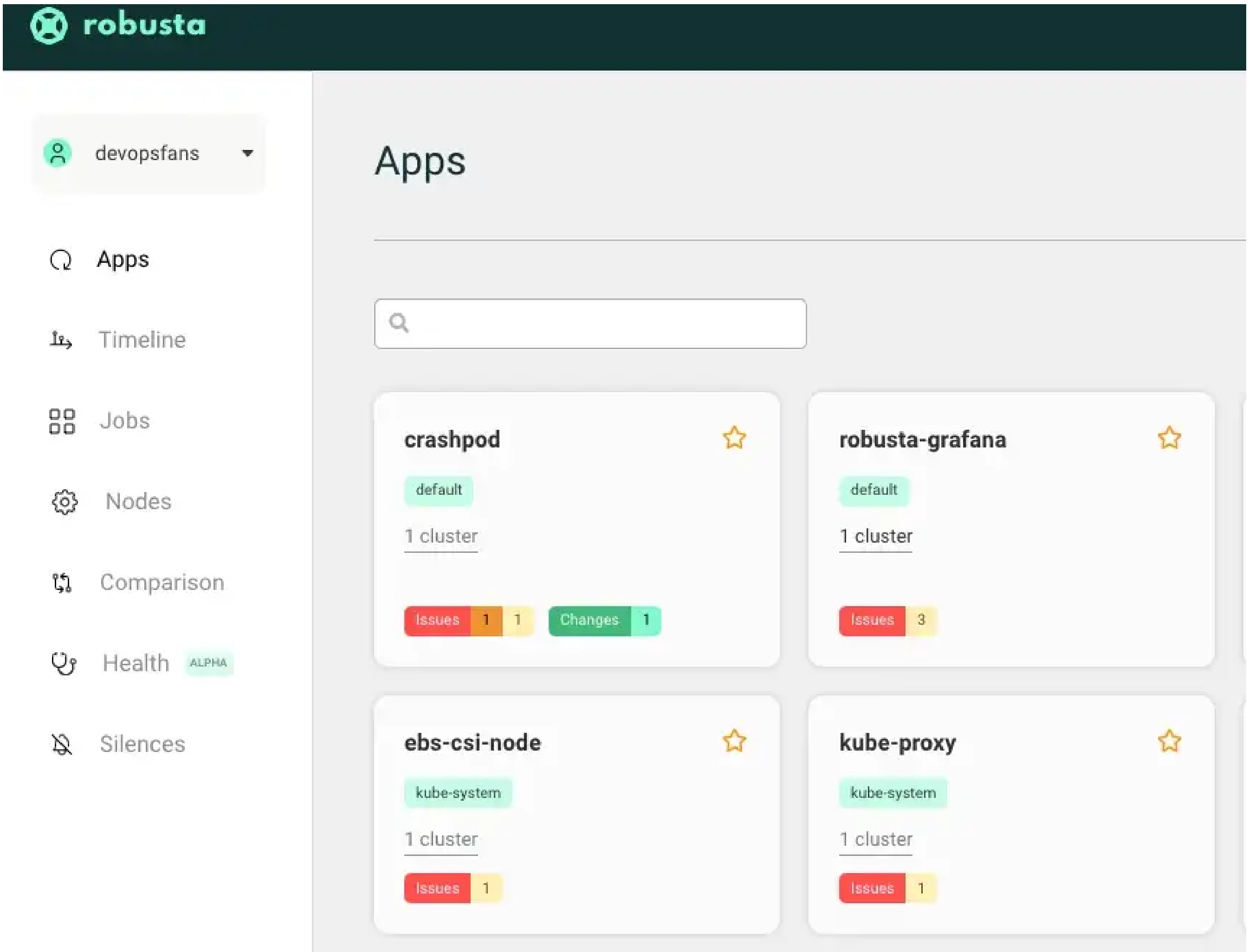
What is Robusta

Robusta (https://home.robusta.dev/) is an open source platform for K8s troubleshooting. Like many other cloud-antive apps, it is installed and managed with Helm3, and sits on top of the monitoring stack (Prometheus, Elasticsearch, etc.) and tells you why alerts occurred and how to fix them.

Robusta comes with the following five parts:

- An automation engine for K8s
- Builtin automations to enrich and fix common K8s alerts
- A manual troubleshooting tool
- An all-in-one bundle with Robusta, the Prometheus Operator, and default K8s alerts.
- A WebUI and operation dashboard

Below is how Robusta Dashboard looks like:



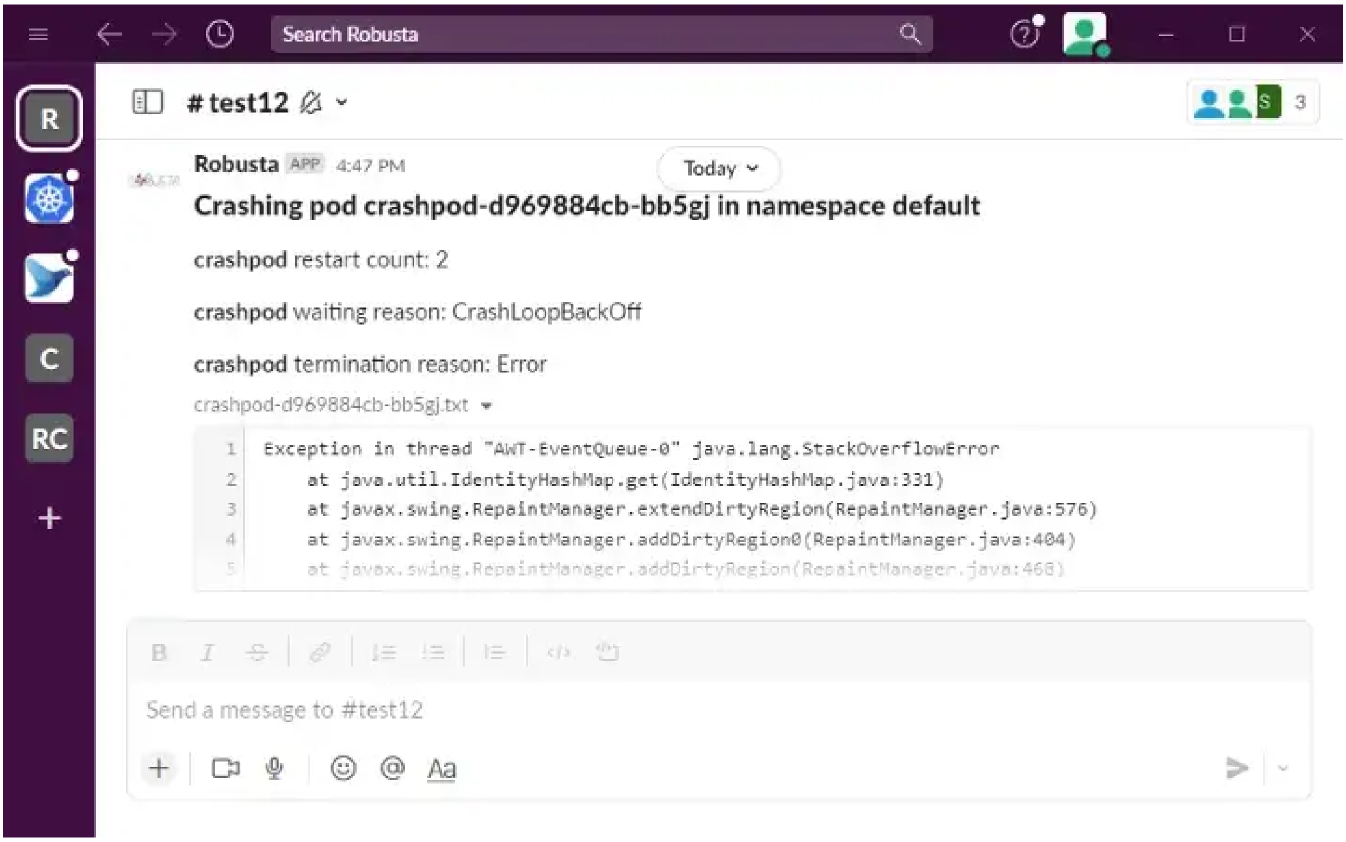
Why Robusta

Robusta is both an automations engine for Kubernetes, and a multi-cluster observability platform. Robusta is commonly used alongside Prometheus, but other tools are supported too.

By listening to all the events in your cluster, Robusta can tell you *why* alerts fired, what happened at the same time, and what you can do about it. It can either improve your existing alerts, or be used to define new alerts triggered by APIServer changes.

Why something happens to your K8s cluster, how to troubleshoot and identify the cause of the problem is usually unclear for K8s administrators, since K8s is such a complicated system. With Robusta integration, it is to get hint about crashing Pods, event correlation, remediate alerts and debug pods.

Crashing Pods:



Robusta APP 4:47 PM

Crashing pod crashpod-d969884cb-bb5gj in namespace default

crashpod restart count: 2

crashpod waiting reason: CrashLoopBackOff

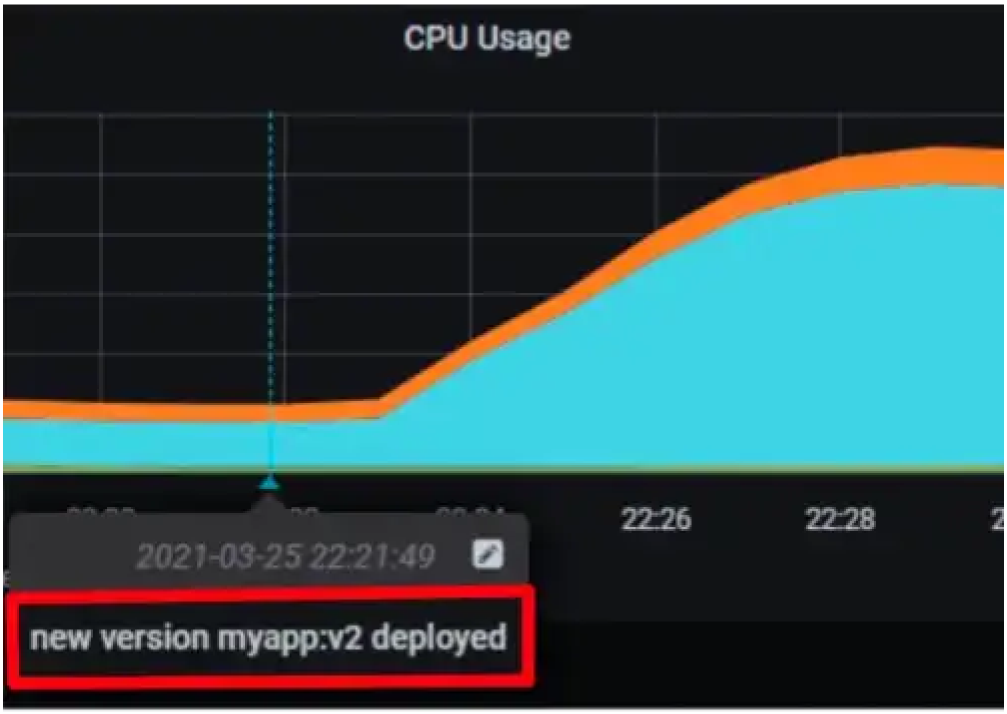
crashpod termination reason: Error

crashpod-d969884cb-bb5gj.txt

```
Exception in thread "AWT-EventQueue-0" java.lang.StackOverflowError
    at java.util.IdentityHashMap.get(IdentityHashMap.java:331)
    at javax.swing.RepaintManager.extendDirtyRegion(RepaintManager.java:576)
    at javax.swing.RepaintManager.addDirtyRegion0(RepaintManager.java:404)
    at javax.swing.RepaintManager.addDirtyRegion(RepaintManager.java:468)
```

Pic from Robusta

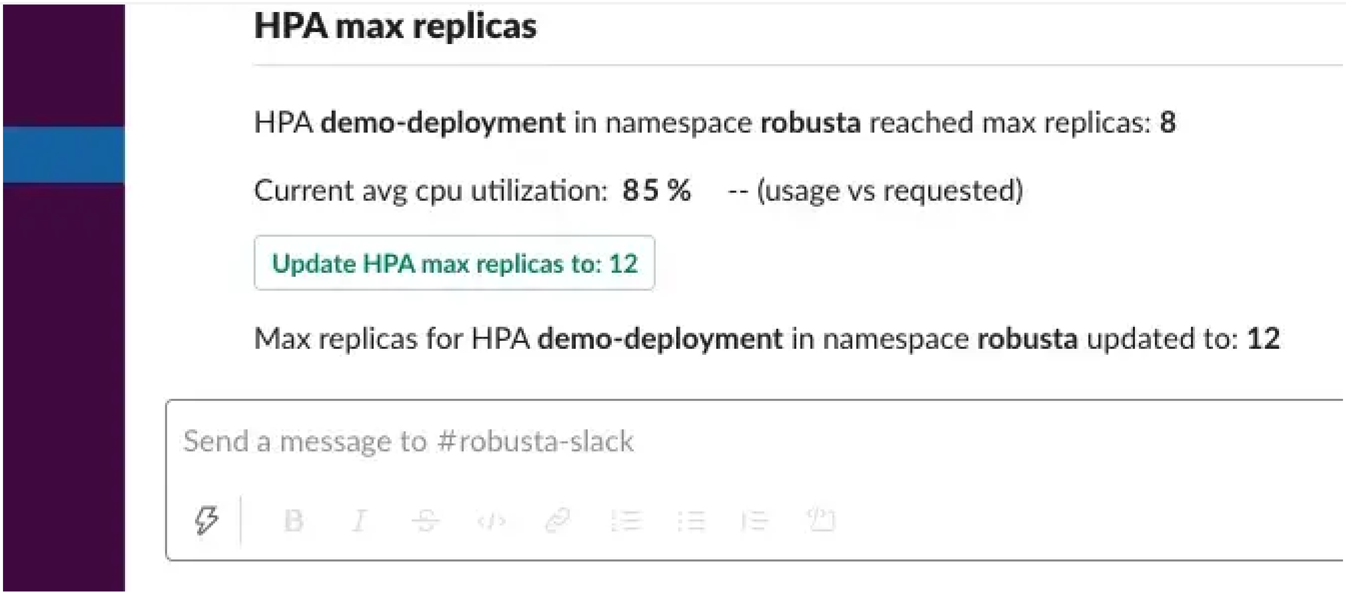
Event Correlation:



2021-03-25 22:21:49

new version myapp:v2 deployed

Pic from Robusta



HPA max replicas

HPA **demo-deployment** in namespace **robusta** reached max replicas: **8**

Current avg cpu utilization: 85 % -- (usage vs requested)

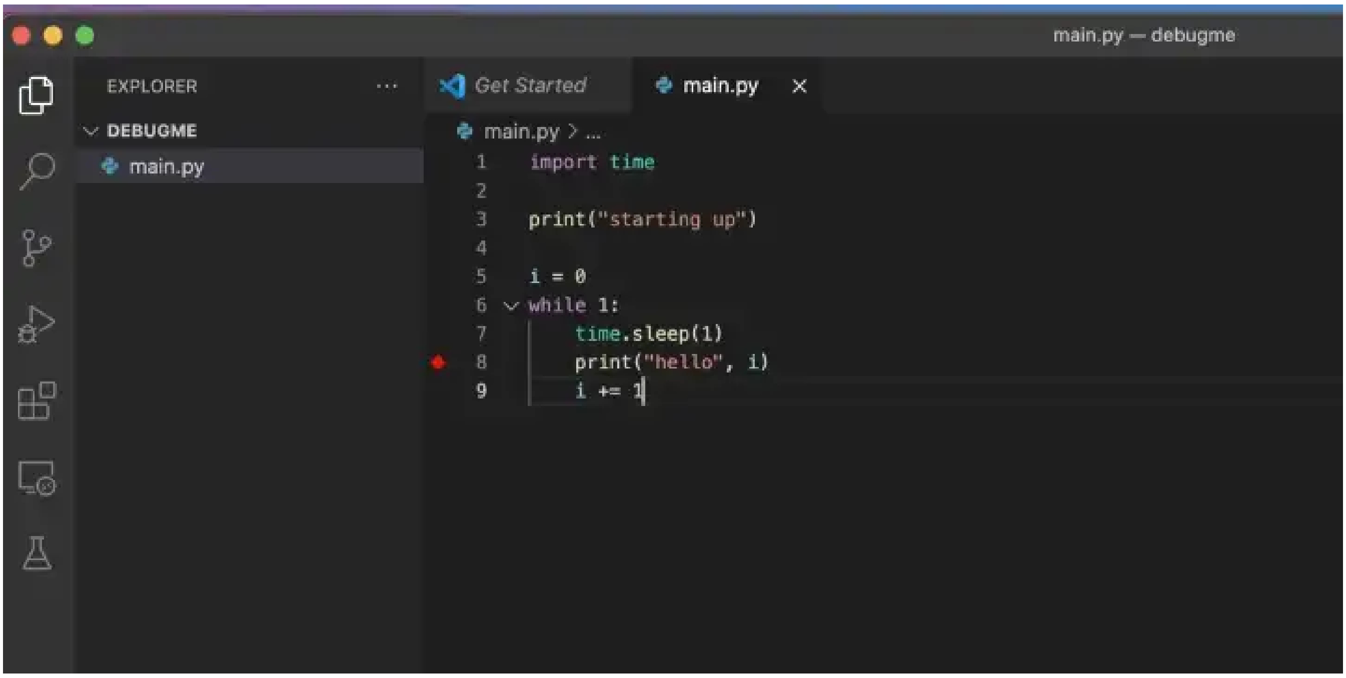
Update HPA max replicas to: 12

Max replicas for HPA **demo-deployment** in namespace **robusta** updated to: **12**

Send a message to #robusta-slack

Pic from Robusta

Debug Pods:

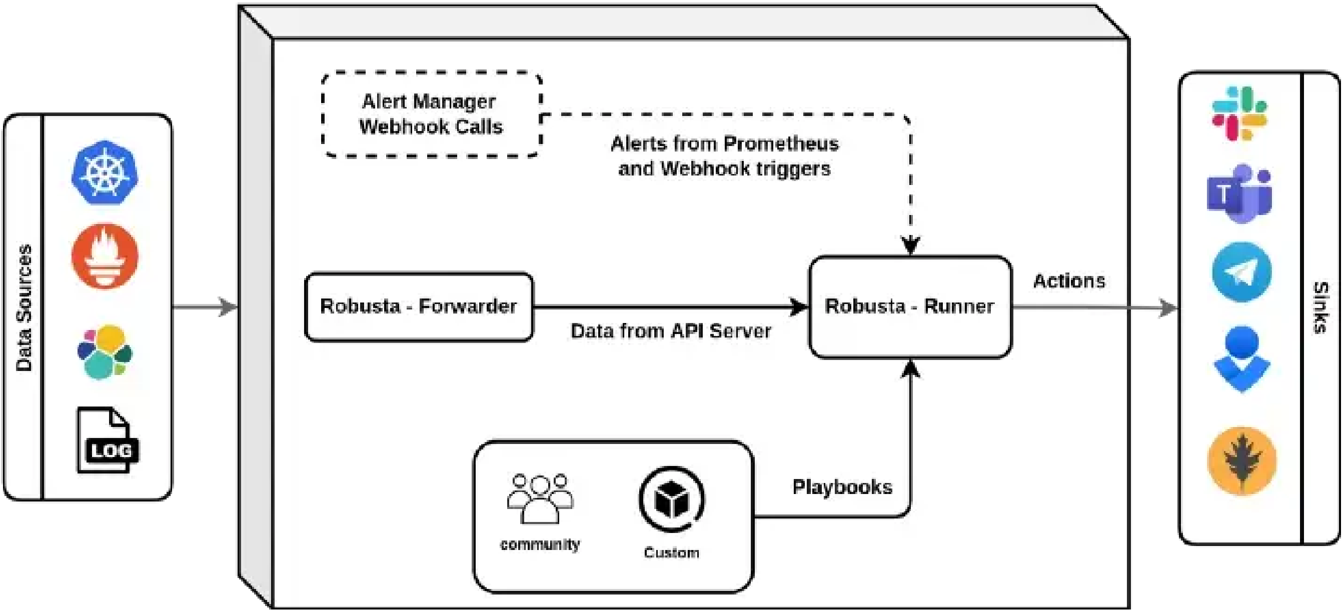


```python
import time

print("starting up")

i = 0
while 1:
    time.sleep(1)
    print("hello", i)
    i += 1
```

Pic from Robusta

Robusta Architecture

The Robusta architecture is:



Pic from robusta.dev

As you can see from the above diagram, the main component of Robusta is the automation engine, which runs in-cluster as two K8s deployments

robusta-forwarder

It connects to the APIServer and monitors K8s changes. Forwards them to robusta-runner.

robusta-runner

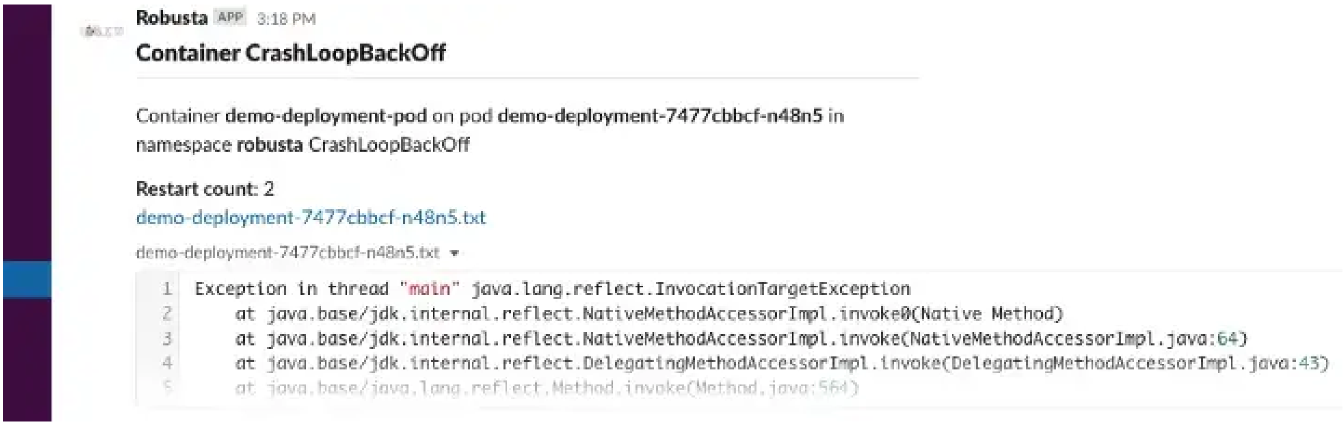
It will executes defined playbooks.

How Robusta Works

Using `CrashingPods` as an example, Robusta's behaviour is defined by rules like the following:

```
triggers:
  - on_prometheus_alert:
      alert_name: KubePodCrashLooping
actions:
  - logs_enricher: {}
sinks:
  - slack
```

In the above example, whenever the `KubePodCrashLooping` alert fires, Robusta will fetch logs from the right pod and attach them to the alert. The result looks like this:



Robusta APP 3:18 PM

Container CrashLoopBackOff

Container **demo-deployment-pod** on pod **demo-deployment-7477cbbcf-n48n5** in namespace **robusta** CrashLoopBackOff

Restart count: 2

demo-deployment-7477cbbcf-n48n5.txt

demo-deployment-7477cbbcf-n48n5.txt

```
Exception in thread "main" java.lang.reflect.InvocationTargetException
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:64)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:564)
```

How to Install Robusta

Since Robusta is managed by Helm, we can use `helm3` to install it.

Install Repo

```
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
```

Generate Configuration File

First we install its cli through Python:

```
$ python3.10 -m venv robusta
$ source robusta/bin/activate
(robusta) $ pip install -U robusta-cli --no-cache
Collecting robusta-cli
Downloading robusta_cli-0.10.10-py3-none-any.whl (223 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 223.8/223.8 kB 30.0 MB/s eta 0:0
Collecting pymsteams<0.2.0,>=0.1.16
  Downloading pymsteams-0.1.16.tar.gz (7.6 kB)
  Preparing metadata (setup.py) ... done
...
Successfully installed PyJWT-2.4.0 appdirs-1.4.4 autopep8-2.0.1 black-21.5b2
cachetools-5.2.1 certifi-2022.12.7 cffi-1.15.1 charset-normalizer-3.0.1
...
ruamel.yaml.clib-0.2.7 six-1.16.0 slack-sdk-3.19.5 tenacity-8.1.0
toml-0.10.2 tomli-2.0.1 typer-0.4.2 typing-extensions-4.4.0 urllib3-1.26.14
 watchgod-0.7 webexteamssdk-1.6.1 websocket-client-1.3.3
```

Then we generate config file:

```
If you haven't installed it yet, Robusta can install a
pre-configured Prometheus.
Would you like to do so? [y/N]: y
Please read and approve our End User License Agreement:
https://api.robusta.dev/eula.html
Do you accept our End User License Agreement? [y/N]: y
Last question! Would you like to help us improve Robusta by sending exception r
Saved configuration to ./generated_values.yaml - save this file for future use.
Finish installing with Helm (see the Robusta docs).
Then login to Robusta UI at https://platform.robusta.dev

By the way, we'll send you some messages later to get feedback.
(We don't store your API key, so we scheduled future messages using Slack's
API)
```

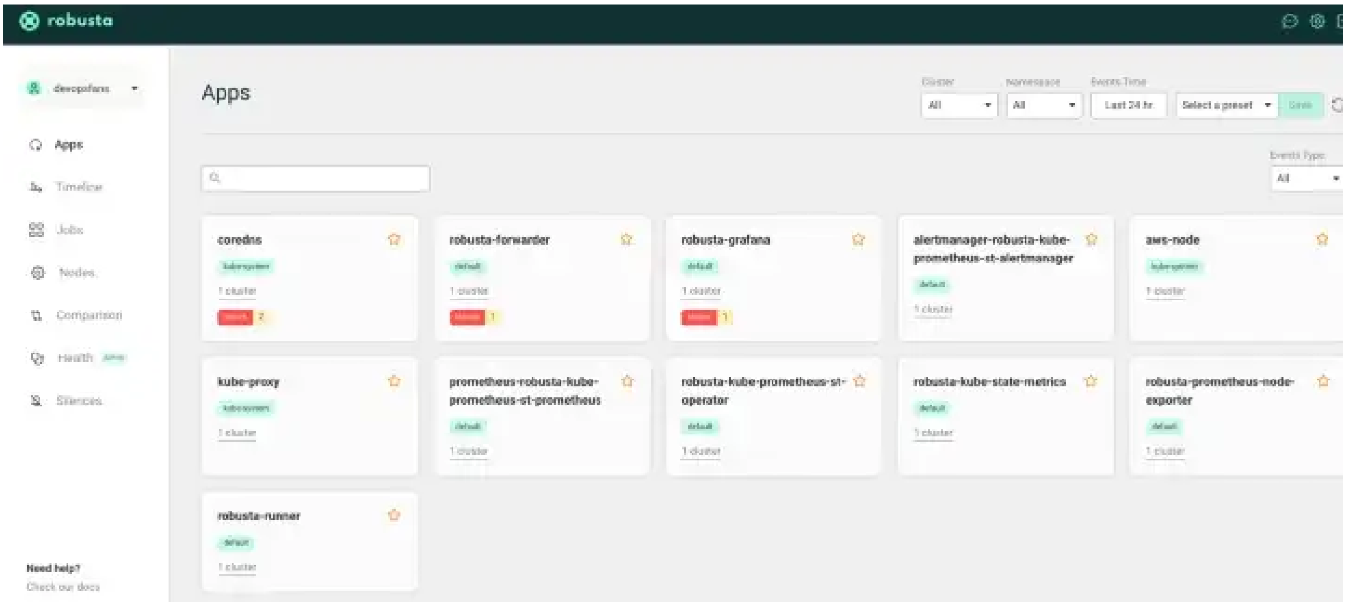
Then we deploy:

```
$ helm install robusta robusta/robusta -f ./generated_values.yaml \
--set clusterName=dev-cluster
```

After that, we verify the two Robusta pods and running with no errors in the logs:

```
$ kubectl get pods -A | grep robusta
$ robusta logs
```

Now you should be able to see your dashboard in https://platform.robusta.dev/



Reference

For more detailed usage of Robusta, please visit https://home.robusta.dev/

Kubernetes Dev Ops Cloud Computing Programming Containers

Enjoy the read? Reward the writer.Beta

Your tip will go to Tony through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Sign up for DevGenius Updates

By Dev Genius

Get the latest news and update from DevGenius publication Take a look.