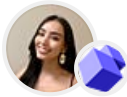


GitOps-Powered Kubernetes Testing Machine: ArgoCD + Testkube



Alejandra Thomas · [Follow](#)

Published in Kubeshop

7 min read · Nov 22, 2023



Listen



Share

Introduction: Challenges to GitOps Cloud Native Testing

One of the major trends in contemporary cloud native application development is the adoption of GitOps; managing the state of your Kubernetes cluster(s) in Git — with all the bells and whistles provided by modern Git platforms like GitHub and GitLab in regard to workflows, auditing, security, tooling, etc. Tools like ArgoCD or Flux are used to do the heavy lifting of keeping your Kubernetes cluster in sync with your Git repository; as soon as difference is detected between Git and your cluster it is deployed to ensure that your repository is the source-of-truth for your runtime environment.

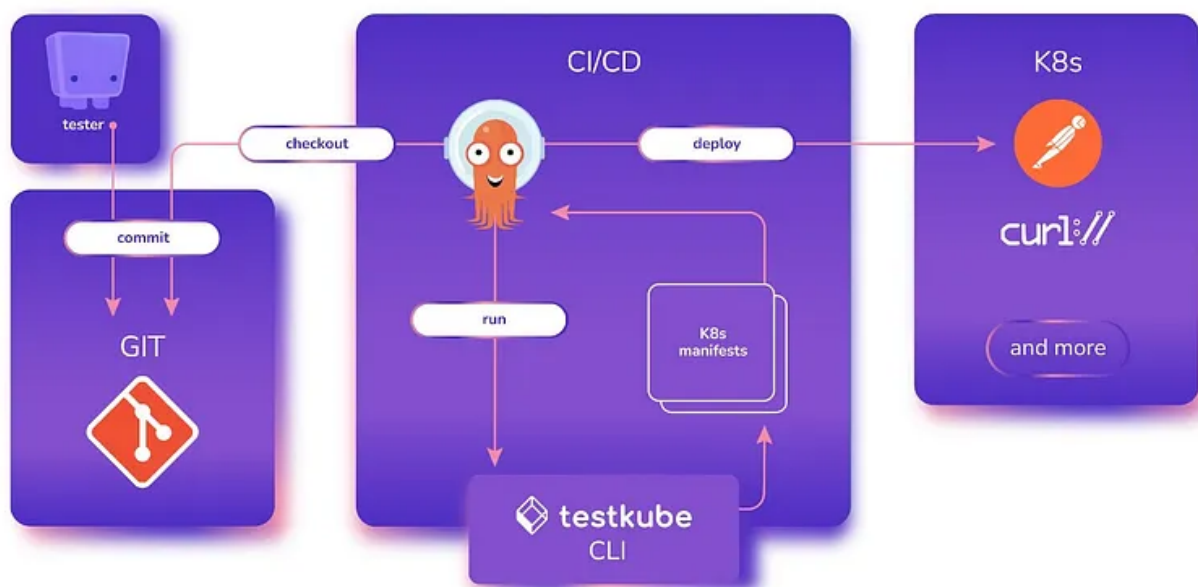
Don't you agree that it's time to move testing and related activities into this paradigm also? Exactly! We at Kubeshop are working hard to provide you with the first GitOps-friendly Cloud-native test orchestration/execution framework — [Testkube](#) — to ensure that your QA efforts align with this new and shiny approach to application configuration and cluster configuration management. Combined with the GitOps approach described above, Testkube will include your test artifacts and application configuration in the state of your cluster and make git the source of truth for these test artifacts.

Benefits of the GitOps approach:

1. Since your tests are included in the state of your cluster you are always able to validate that your application components/services work as required.

2. Since tests are executed from inside your cluster there is no need to expose services under test externally purely for the purpose of being able to test them.
3. Tests in your cluster are always in sync with the external tooling used for authoring
4. Test execution is not strictly tied to CI but can also be triggered manually for ad-hoc validations or via internal triggers (Kubernetes events)
5. You can leverage all your existing test automation assets from Postman, or Cypress (even for end-to-end testing), or ... through executor plugins.

Conceptually, this can be illustrated as follows:



GitOps Tutorial

Enough talk — let's see this in action — here comes a step-by-step walkthrough to get this in place for the automated application deployment and execution of Postman collections in a local Minikube cluster to test.

Let's start with setting things up for our GitOps-powered testing machine!

Pre-Requisites for GitOps Testing

You can follow the minikube installation for your operating system [here](#).

Follow the [ArgoCD installation guide](#).

Note: For step 3 “Access The Argo CD API Server”, choose the “Port Forwarding” method, as that is the easiest way to connect to it with a Minikube cluster.

Follow the installation guide for Testkube [here](#). Make sure to install the CLI client and the components in your cluster.

Set up “Hello Kubernetes” application and tests

1. Install a “Hello Kubernetes!” application in your cluster

We will create a YAML file for a simple “Hello Kubernetes” application that we will then create our integration tests against.

```
apiVersion: v1
kind: Service
metadata:
  name: hello-kubernetes-service
spec:
  ports:
    - name: http
      port: 80
      targetPort: 8080
  selector:
    app: hello-kubernetes
  - -
```

And deploy the *Hello Kubernetes* deployment with:

```
kubectl apply -f hello-kubernetes.yaml
```

You can test that your application has been correctly installed by running:

```
minikube service hello-kubernetes-service
```

2. Set up a Git Repository containing some Postman collections

We are going to use tests created by Postman and exported in a *Postman collections* file.

We can upload this to the same Git Repository as our application, but in practice the repository could be the same repository hosting the application or it could also be in a separate repository where you manage all your test artifacts.

So let's create our **hello-kubernetes.json** and push it to the repository:

```
{
  "info": {
    "_postman_id": "02c90123-318f-4680-8bc2-640adabb45e8",
    "name": "New Collection",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection."
  },
  "item": [
    {
      "name": "hello-world test",
      "event": [
        {
          "listen": "test",
          "script": {
            "exec": [
              "pm.test(\"Body matches string\", () => {",
              "    pm.expect(pm.response.text()).to.contain(\"Hello Kubernetes\"",
              "})",
              "",
              "pm.test(\"Body matches string\", () => {",
              "    pm.expect(pm.response.status).to.equal(\"OK\")",
              "})"
            ],
            "type": "text/javascript"
          }
        }
      ],
      "request": {
        "method": "GET",
        "header": [],
        "url": {
          "raw": "http://hello-kubernetes-service.default",
          "protocol": "http",
          "host": [
            "hello-kubernetes-service",
            "default"
          ]
        }
      },
      "response": []
    }
  ]
}
```

```
}  
]  
}
```

You can see an example of how the repository should look like [here](#).

Configure ArgoCD to work with Testkube

1. Configure ArgoCD to use the Testkube plugin

To get ArgoCD to use Testkube, we need to add Testkube as a plugin. To do so, please nest the plugin config file in a ConfigMap manifest under the ***plugin.yaml*** key.

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: argocd-cm-plugin  
  namespace: argocd  
data:  
  plugin.yaml: |  
    apiVersion: argoproj.io/v1alpha1  
    kind: ConfigManagementPlugin  
    metadata:  
      name: testkube  
    spec:  
      version: v1.0  
      generate:  
        command: [bash, -c]  
        args:  
          - |  
            testkube generate tests-crds .
```

And apply it with the following command:

```
kubectl apply -f argocd-plugins.yaml
```

As you can see here, we're using the command `testkube generate tests-crds` which creates the Custom Resources (manifests) that ArgoCD will then add to our cluster.

To install a plugin, patch *argocd-repo-server* deployment to run the plugin container as a sidecar.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: argocd-repo-server
spec:
  template:
    spec:
      containers:
      - name: testkube
        command: [/var/run/argocd/argocd-cmp-server]
        image: kubeshop/testkube-argocd:latest
        securityContext:
          runAsNonRoot: true
          runAsUser: 999
        volumeMounts:
          - mountPath: /var/run/argocd
            name: var-files
          - mountPath: /home/argocd/cmp-server/plugins
            name: plugins
          - mountPath: /home/argocd/cmp-server/config/plugin.yaml
            subPath: plugin.yaml
            name: argocd-cm-plugin
          - mountPath: /tmp
            name: cmp-tmp
      volumes:
      - configMap:
          name: argocd-cm-plugin
          name: argocd-cm-plugin
      - emptyDir: {}
        name: cmp-tmp
```

Apply the patch with the command:

```
kubectl patch deployments.apps -n argocd-repo-server --patch-file deployment.yaml
```

Create the file that will contain the ArgoCD application:

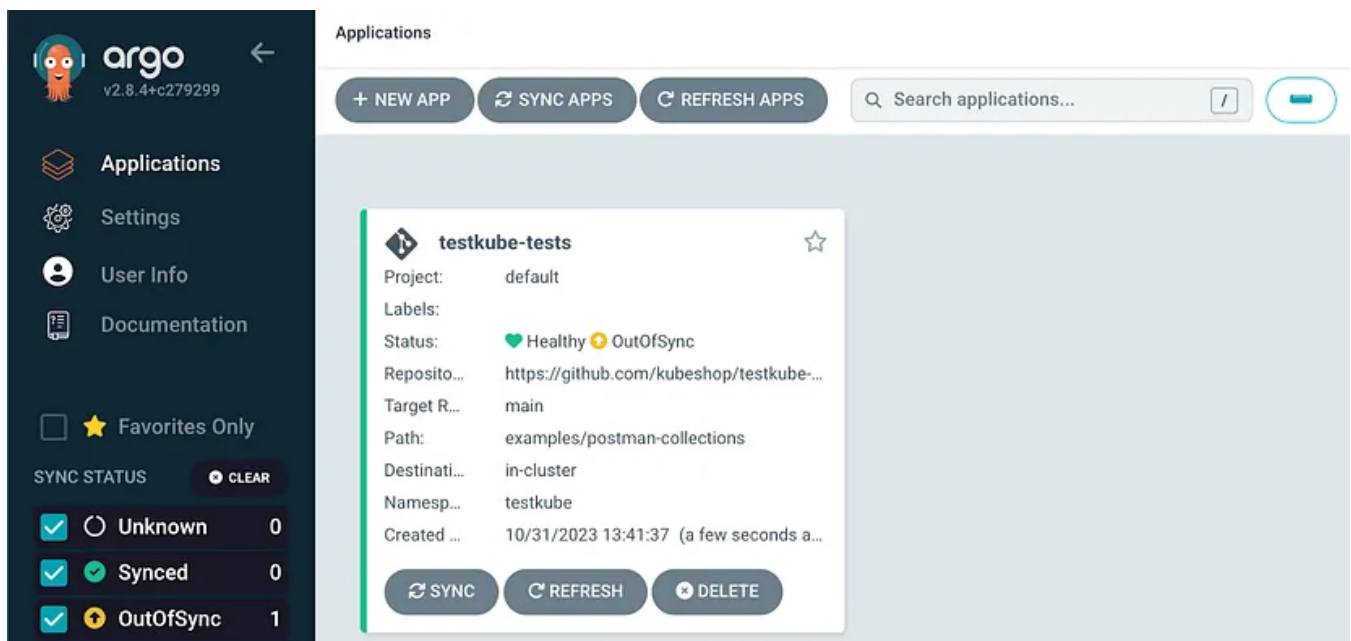
```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: testkube-tests
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/USERNAME/testkube-argocd.git
    targetRevision: HEAD
    path: postman-collections
    plugin:
      name: "testkube-v1.0"
  destination:
    server: https://kubernetes.default.svc
    namespace: testkube
```

Notice that we have defined `path: postman-collections` which is the test folder with our Postman collections from the steps earlier. With Testkube you can use multiple test executors like **curl** for example, so it is convenient to have a folder for each. We have also defined the **.destination.namespace** to be **testkube**, which is where the tests should be deployed in our cluster.

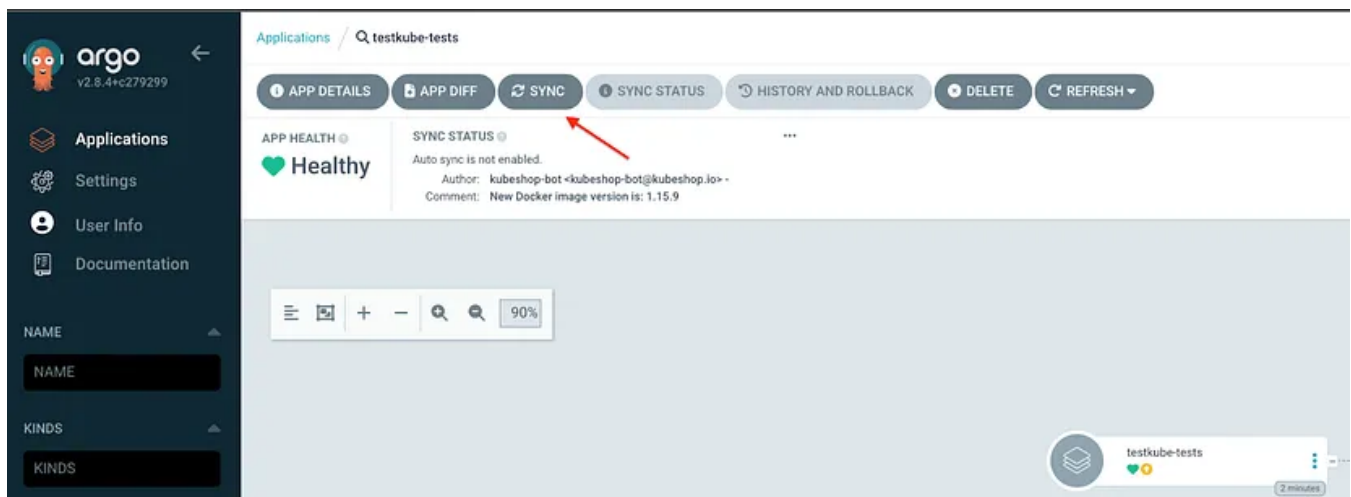
Now let's create the application with:

```
kubectl apply -f testkube-application.yaml
```

On ArgoCD's dashboard, we will now see the newly created application. Let's click to get into it and sync our tests.



And now click on **Sync** to see your tests created.



And voilà, there's our test collection created and managed by ArgoCD with every new test created and updated in the Github repository containing the tests!

APP DETAILS

APP DIFF

SYNC

SYNC STATUS

HISTORY AND ROLLBACK

DELETE

REFRESH

APP HEALTH Healthy

SYNC STATUS Auto sync is not enabled.
Author: ypoplavs <45286051+ypoplavs@users.noreply.github...
Comment: add another example (#9)

testkube-tests

hello-kubernetes

petstore-postman-collection

todo-postman-collection

Run your ArgoCD tests!

1. Run ad-hoc tests from the CLI

Now that we're all set — let's try some ad-hoc test execution using Testkube's CLI

List the tests in your cluster with:

```
testkube get tests
```

You should see your deployed test artifacts

```
> testkube get tests
Context: (1.15.9) Namespace: testkube
-----
NAME                | DESCRIPTION | TYPE           | CREATED                | LABELS                                     | SCHEDULE | STATUS | EXECUTION ID
-----
hello-kubernetes    |             | postman/collection | 2023-10-31 11:38:34 +0000 UTC | app.kubernetes.io/instance=testkube-tests |          |       |
petstore-postman-collection |             | postman/collection | 2023-10-31 11:38:34 +0000 UTC | app.kubernetes.io/instance=testkube-tests |          |       |
todo-postman-collection |             | postman/collection | 2023-10-31 11:38:34 +0000 UTC | app.kubernetes.io/instance=testkube-tests |          |       |
```

To run those tests execute the following command:

```
testkube run test hello-kubernetes
```


The test execution will start in the background, you now need to copy the command from the image below to check the result of the execution of the test

```
> testkube run test hello-kubernetes

Context: (1.15.9)  Namespace: testkube
-----
Type:      postman/collection
Name:      hello-kubernetes
Execution ID: 654119696b7a59b77919c507
Execution name: hello-kubernetes-3
Execution number: 3
Status:    running
Start time: 2023-10-31 15:12:41.195255592 +0000 UTC
End time:  0001-01-01 00:00:00 +0000 UTC
Duration:

Test execution started
Watch test execution until complete:
$ kubectl testkube watch execution hello-kubernetes-3

Use following command to get test execution details:
$ kubectl testkube get execution hello-kubernetes-3
```

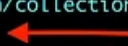


```
testkube get execution EXECUTION_ID
```

And you should see that the tests have run successfully, just like in the image below.

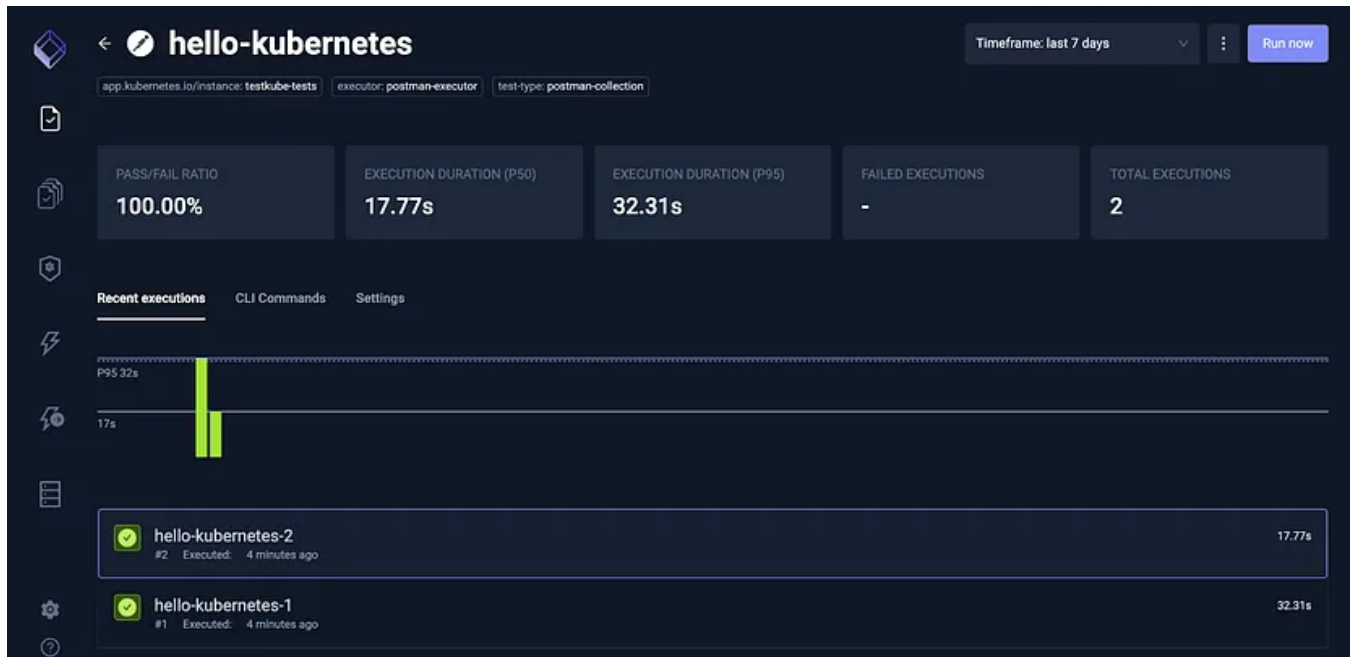
```
> testkube get execution hello-kubernetes-1

Context: (1.15.9)  Namespace: testkube
-----
ID:      6540f4f66b7a59b77919c503
Name:    hello-kubernetes-1
Number:  1
Test name: hello-kubernetes
Type:    postman/collection
Status:  passed
Start time: 2023-10-31 12:37:10.547 +0000 UTC
End time:  2023-10-31 12:37:42.862 +0000 UTC
Duration:  00:00:32
Running context:
Type:  user-cli
Context:
Labels:  app.kubernetes.io/instance=testkube-tests
Command: newman
Args:    run <runPath> -e <envFile> --reporters cli,json --reporter-json-export <reportFile>
```



You can also see the results of your tests in a nice dashboard. Just open the Testkube dashboard with the following command

```
testkube dashboard
```



And you will be able to see the results of the execution in the *Executions* tab as seen in the image below.

GitOps Takeaways

Once fully realized — using GitOps for testing of Kubernetes applications as described above provides a powerful alternative to a more traditional approach where orchestration is tied to your current CI/CD tooling and not closely aligned with the lifecycle of Kubernetes applications.

Would love to get your thoughts on the above approach — over-engineering done right? Waste of time? Let us know!

Check Testkube on GitHub — and let us know if you're missing something we should be adding to make your k8s resource testing easier.

Get started by [signing into Testkube now](#).

Thank you!

Originally published at <https://testkube.io>.

Kubernetes

DevOps

Testing

ArgoCD

GitOps



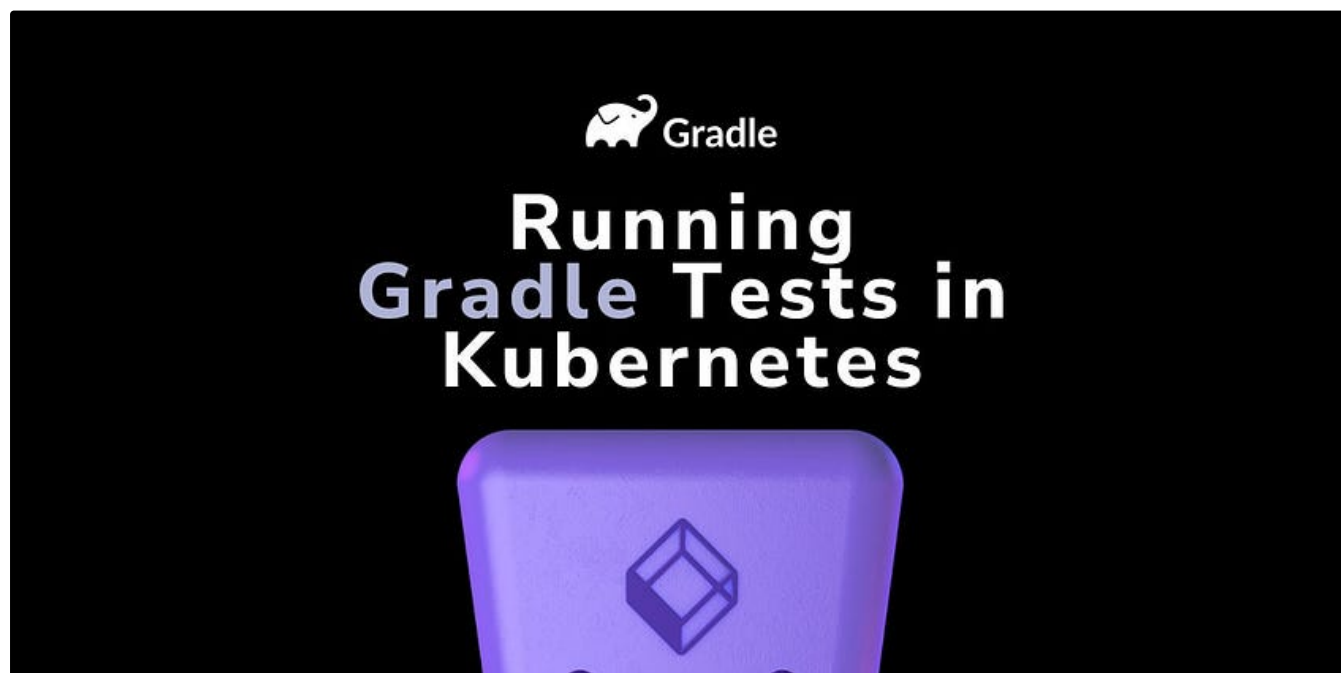
Follow


Written by Alejandra Thomas

226 Followers · Editor for Kubeshop

Developer Advocate & Web Developer — Talks about cloud, back-end, and web stuff.

More from Alejandra Thomas and Kubeshop




 Alejandra Thomas in Kubeshop

Running Gradle Tests in Kubernetes

As Gradle remains a popular build tool for Java applications, and with Kubernetes establishing itself as the go-to orchestration platform...

4 min read · Jan 25, 2024

 34





Error occurred for Pod **default/kube-prometheus-stack-grafana-5b4c5bf6c-qz57** in **botkube-lab** cluster

Messages:

- Readiness probe failed: Get "<http://172.0.90.17:3000/api/health>": context deadline exceeded (Client.Timeout exceeded while awaiting headers)

2023-02-20 12:20:13 PM

Run command...

`kubectl logs pod/kube-prometheus-stack-grafana-5b4c5bf6c-qz57 -n default` on **botkube-lab** by Automation "Show logs on error"

```
Defaulted container "grafana-sc-dashboard" out of: grafana-sc-dashboard, grafana-sc-datasources, grafana
```

 Maria in Kubeshop

5 Essential K8s Tasks to Automate

Kubernetes has greatly changed the DevOps space and how we deploy applications. While it offers many powerful features, it also introduces...

Open in app ↗

Sign up

Sign in

 Medium

 Search



tkube APP 12:09 PMSource: **prometheus**Error Name: **KubeStatefulSetReplicasMismatch**State: **pending**

Description:

StatefulSet test/postgresql-12-1675422493 has not matched the expected number of replicas for longer than 15 minutes.

Source: **prometheus**Error Name: **KubePodNotReady**State: **pending**

Description:

Pod test/postgresql-12-1675422493-0 has been in a non-ready state for longer than 15 minutes.

Source: **prometheus**Error Name: **KubeContainerWaiting**State: **pending**

Description:

Pod test/postgresql-12-1675422493-0 in namespace test on container postgresql has been in waiting state for longer than 1 hour



Maria in Kubeshop

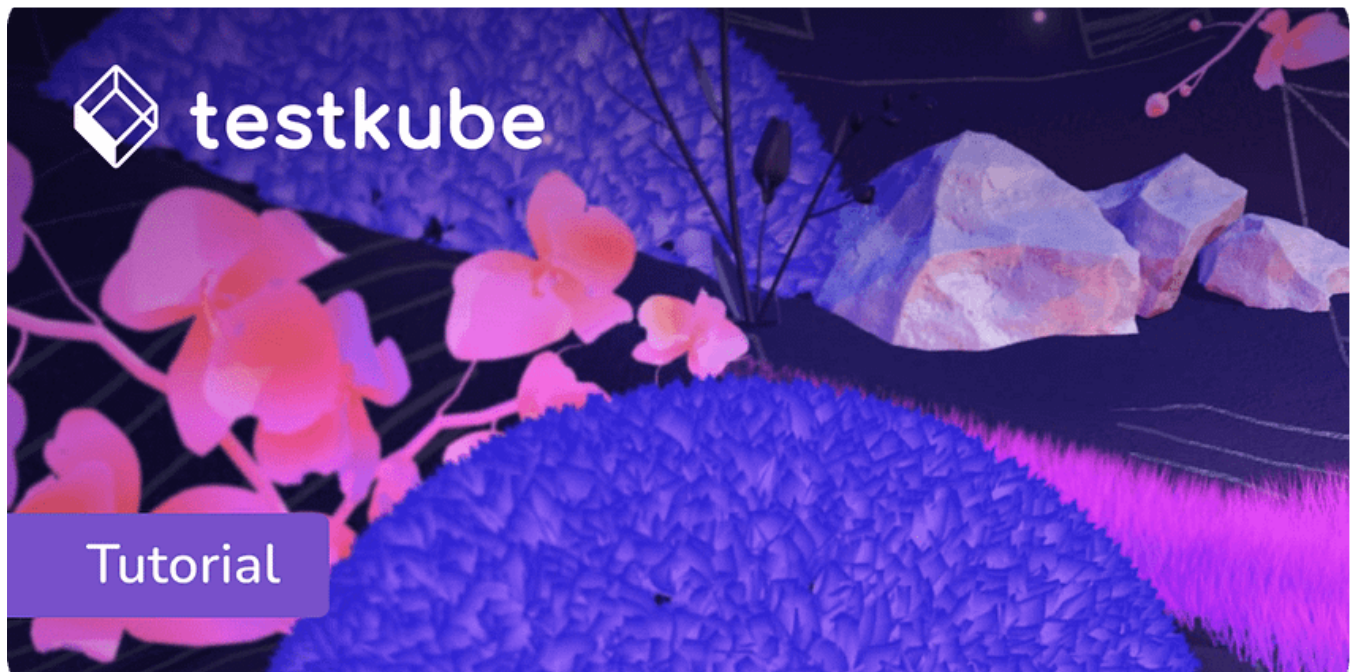
5 Essential K8s Troubleshooting + Monitoring Tasks with Botkube

If you're curious about the exciting possibilities of using Kubernetes in your chat platform, you've come to the right place. With Botkube...

7 min read · Jan 18, 2024



2



Alejandra Thomas in Kubeshop

Uniting Test Automation Tools With Testkube's Container Executors