

2.0 Installation


Contents of the current section:

- [Overview](#)
- [Requirements and tools](#)
- [Infrastructure deployment](#)
 - [1. Login to the terraform cloud.](#)
 - [2. Select VCS provider.](#)
 - [3. Choose repository](#)
 - [4. Open Advanced options](#)
 - [5. Define folder with terraform code](#)
 - [6. Define branch name](#)
 - [7. Create workspace](#)
 - [8. Define required variables](#)
 - [9. Start plan](#)
 - [10. Apply](#)
- [Application deployment](#)
 - [1. Create secrets and variables](#)
 - [2. Trigger CI pipeline](#)
 - [3. Wait for the CI process to finish](#)
 - [4. Get argocd default admin password](#)
 - [5. Login to argo CD and check sync](#)

Overview

In this section we will cover up installation process from scratch. Stack deployment will consists of two parts infrastructure and application

Requirements and tools

1. terraform cli
2. AWS account (where everything will be deployed)
3. AWS account where our host_zone is located
4. domain (example.com)
5. Cloud terraform account (app.terraform.io)
6. Repo access ( [GitHub - tntk-io/tntk-infra](https://github.com/tntk-io/tntk-infra))

Infrastructure deployment

To deploy our terraform code first of all we need to create account on app.terraform.io and create organization and workspace in my example i created **opex-apex** organization

1. Login to the terraform cloud.



a workspace to get started
managing infrastructure.

are the building blocks of Terraform Cloud
manages a collection of infrastructure.

work with CLI, API, and version control to
manage and track the state of resources.

Create a workspace

Create a workspace

View our Getting Started guide →

Already use Terraform?

Migrate your state →

after that you need to create workspace. We need to go on **workspaces** → **new workspace** → **create workspace** click on then Version control workflow and we have to connect it with our github repository where our terraform code is located in our case it will be `tk-io/tntk-infra` but you should you your own repository.

2. Select VCS provider.


Create a new Workspace

Workspaces determine how Terraform Cloud organizes infrastructure. A workspace contains your Terraform configuration (infrastructure as code), shared variable values, your current and historical Terraform state, and run logs. [Learn more](#) about workspaces in Terraform Cloud.

1 Choose Type 2 **Connect to VCS** 3 Choose a repository 4 Configure settings

Connect to a version control provider

Choose the version control provider that hosts the Terraform configuration for this workspace.

 GitHub 

[Connect to a different VCS](#)

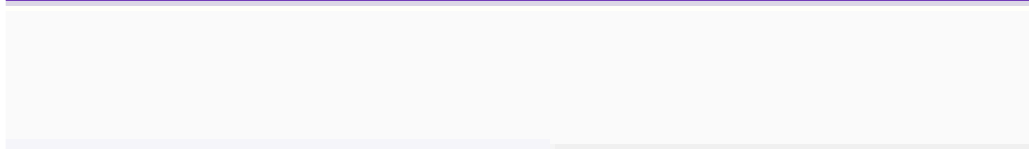
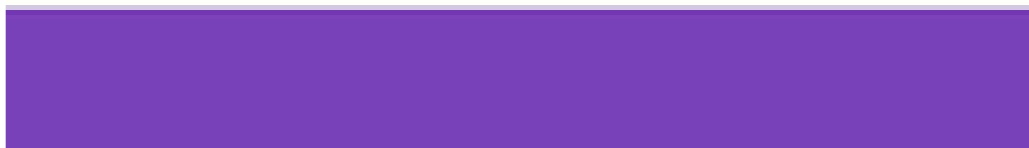
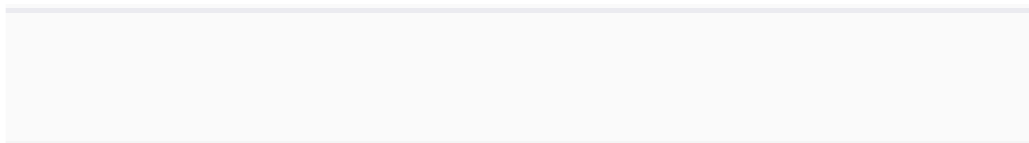
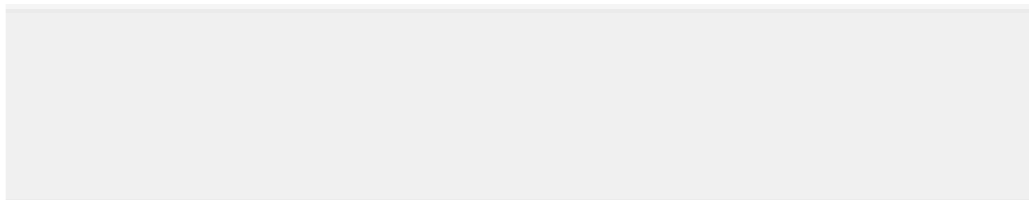
3. Choose repository



Confirm changes

'll watch this for commits and pull requests

get started.




structure :

4. Open Advanced options

Configure Settings

Workspace Name

tntk-infra

The name of your workspace is unique and used in tools, routing, and UI. Dashes, underscores, and alphanumeric characters are permitted. [Learn more about naming workspaces](#) .


Project

✓ Default Project

Every workspace must belong to a single project. Projects must be named uniquely within an organization. Workspaces may be moved between projects at any time from the workspace list or settings. [Learn more about projects](#) .

Description (Optional)

Workspace description

▼ Advanced options 

< Previous

Cancel

Create

5. Define folder with terraform code

Workspace Settings

Terraform Working Directory

The directory that Terraform will execute within. This defaults to the root of your repository and is typically set to a subdirectory matching the environment when multiple environments exist within the same repository.

Terraform will change into the `prod` directory prior to executing any operation. Any modules utilized can be referenced outside of this directory.

6. Define branch name

VCS Triggers

Automatic Run Triggering

Choose when runs should be triggered by VCS changes.

☒ **Always trigger runs**

☐ **Only trigger runs when files in specified paths change**
Supports either glob patterns or prefixes.

☐ **Trigger runs when a git tag is published**
Git tags allow you to manage releases.

VCS branch

multi-stage

The branch from which to import new versions. This defaults to the value your version control provides as the default branch for this repository.

or leave it empty if you are using only one default branch

7. Create workspace

Pull Requests

- ☒ **Automatic speculative plans**
Trigger speculative plans for pull requests to this repository.

Other Settings

- ☐ **Include submodules on clone**
Checking this box will perform a recursive clone of your repositories submodules, making them available in the resulting slug containing your Terraform configuration. Recursive clone is performed with `--depth 1`.

 **Create workspace** **Cancel**

8. Define required variables

Workspace variables (12)

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set precedence [↗](#).

Key	Value	Category	
aws_region	us-east-1	terraform	...
base_domain	XXXXXXXXXX	terraform	...
cd_project_repo	XXXXXXXXXXXXXXXXXXXX	terraform	...
ci_project_repo	XXXXXXXXXXXXXXXXXXXX	terraform	...
datadog_api_key SENSITIVE	Sensitive - write only	terraform	...
datadog_application_key SENSITIVE	Sensitive - write only	terraform	...
datadog_region	us5.datadoghq.com	terraform	...
id_rsa SENSITIVE	Sensitive - write only	terraform	...
registrationToken SENSITIVE	Sensitive - write only	terraform	...
tag_env	prod	terraform	...
AWS_ACCESS_KEY_ID iam user "admin1" SENSITIVE	Sensitive - write only	env	...
AWS_SECRET_ACCESS_KEY iam user "admin1" SENSITIVE	Sensitive - write only	env	...

+ Add variable

Don't forget to define env variables for AWS, if you are using SSO add the `AWS_SESSION_TOKEN`:

[illegible]

9. Start plan

Start your first plan

After you configure any required input variables, start your first plan.

[Start new plan](#)

10. Apply

After planning we can **confirm & apply** – it will execute the changes defined by our *Terraform* configuration to create, update, or destroy resources. It will take about 30 min to create all resources. **In some cases you could get error on helm creation step, to resolve it you**

just need to start new run and error will go away

Needs Confirmation: Check the plan and confirm to apply it, or discard the run.

Confirm & Apply

Discard Run

Add Comment

✓ Plan finished a minute ago

Resources: 128 to add, 0 to change, 0 to destroy

✓ Cost estimation finished a minute ago

Resources: 3 of 35 estimated · \$710.18/mo · +\$710.18

🕒 Apply pending

⚡ Please review the following changes before continuing:

To create

+ 128

Choosing "Confirm & Apply" below will execute the above changes.
Please [review the plan output](#) before proceeding.

Confirm & Apply

Discard Run

Add Comment

Application deployment

After deploying infrastructure we moving to application section. In this section we will deploy our application to our infrastructure.

1. Create secrets and variables

Before starting the CI, you need to create the necessary variables and secrets. You can do this manually through the UI or use the GitHub API ([REST API endpoints for GitHub Actions Secrets - GitHub Docs](#)). You can also do this via GitHub CLI after performing authentication:

Variables example:

```
1 gh variable set ACCOUNT_ID --repo tntk-io/tntk-ci --body "012345678901"
2 gh variable set AWS_REGION --repo tntk-io/tntk-ci --body "us-east-1"
3 gh variable set BASE_DOMAIN --repo tntk-io/tntk-ci --body "your-domain.com"
4 gh variable set APPLICATION_NAME --repo tntk-io/tntk-ci --body "demoapp"
5 gh variable set APPLICATION_NAMESPACE --repo tntk-io/tntk-ci --body "application"
6 gh variable set CD_DESTINATION_OWNER --repo tntk-io/tntk-ci --body "tntk-io"
7 gh variable set CD_PROJECT --repo tntk-io/tntk-ci --body "tntk-cd"
```

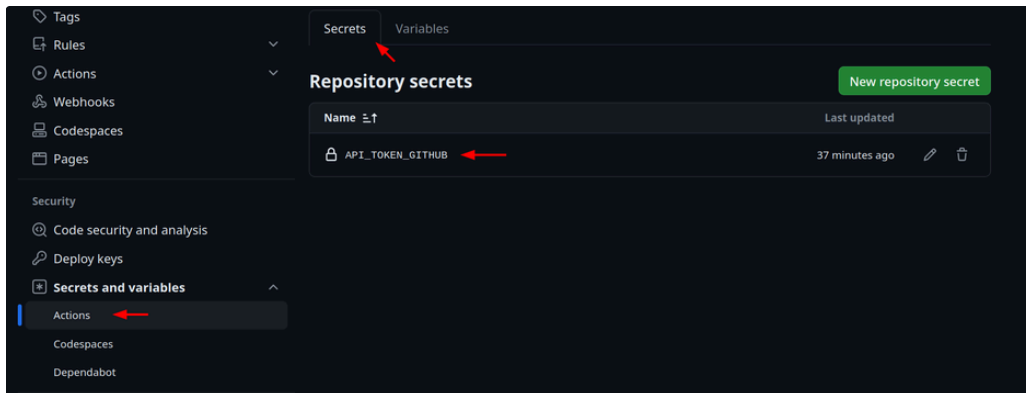
The data provided in the variables above are for example purposes only. Use your own values.

Also you have to create **secret** named `API_TOKEN_GITHUB` which will be used on the "k8s_manifest_storing" stage:

```
1 gh secret set API_TOKEN_GITHUB --repo tntk-io/tntk-ci --body "ghp_TEST_SAMPLEdashgfwqohgohr21ihfih12f9hf1"
```

UI example:

Repo > Settings > Secrets and variables > Actions > Secrets/Variables:

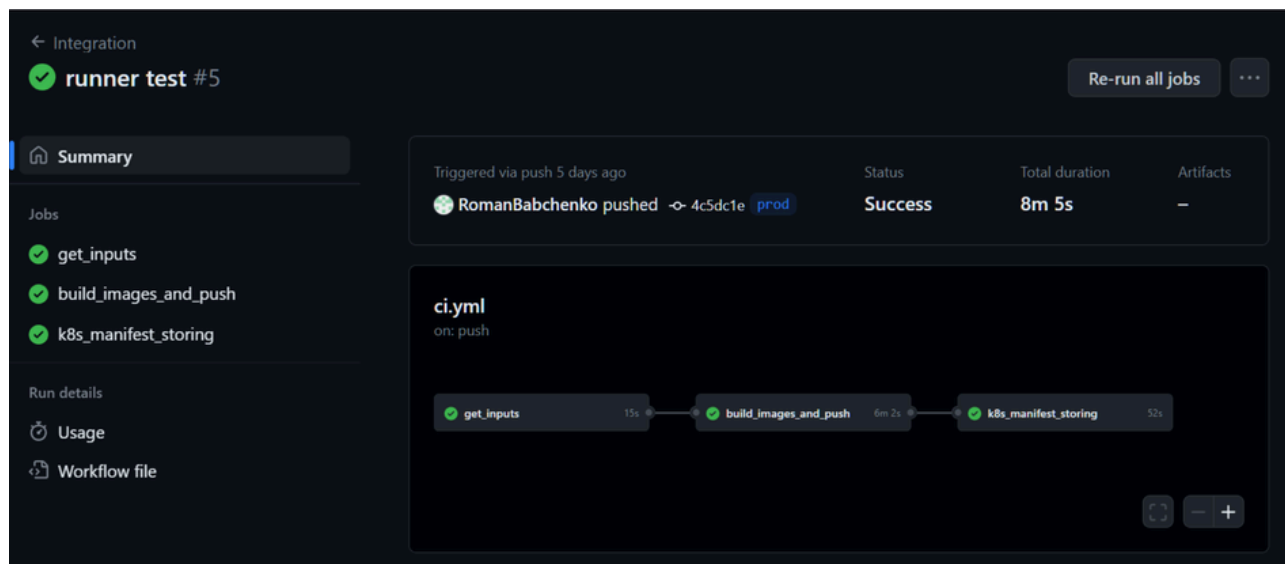


2. Trigger CI pipeline

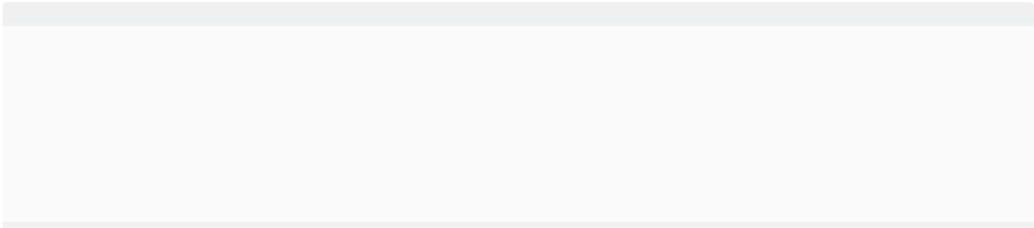
Our integration process depends on commit SHA and the easiest way to start new build is make a empty commit to the CI branch. Trigger CI pipeline by pushing new commit to project src repo:

```
1 git commit --allow-empty -m "test empty commit"
```

3. Wait for the CI process to finish



4. Get argocd default admin password



status
Active

Add-ons

Authentication

Logging

Secrets (6)

Information, such as passwords, OAuth tokens, and secrets

▼

🔍

Filter Secrets by property or value

admin-secret

applications-secret

:.v1.argocd-apps.v1

:.v1.argocd.v1

AWS cli:

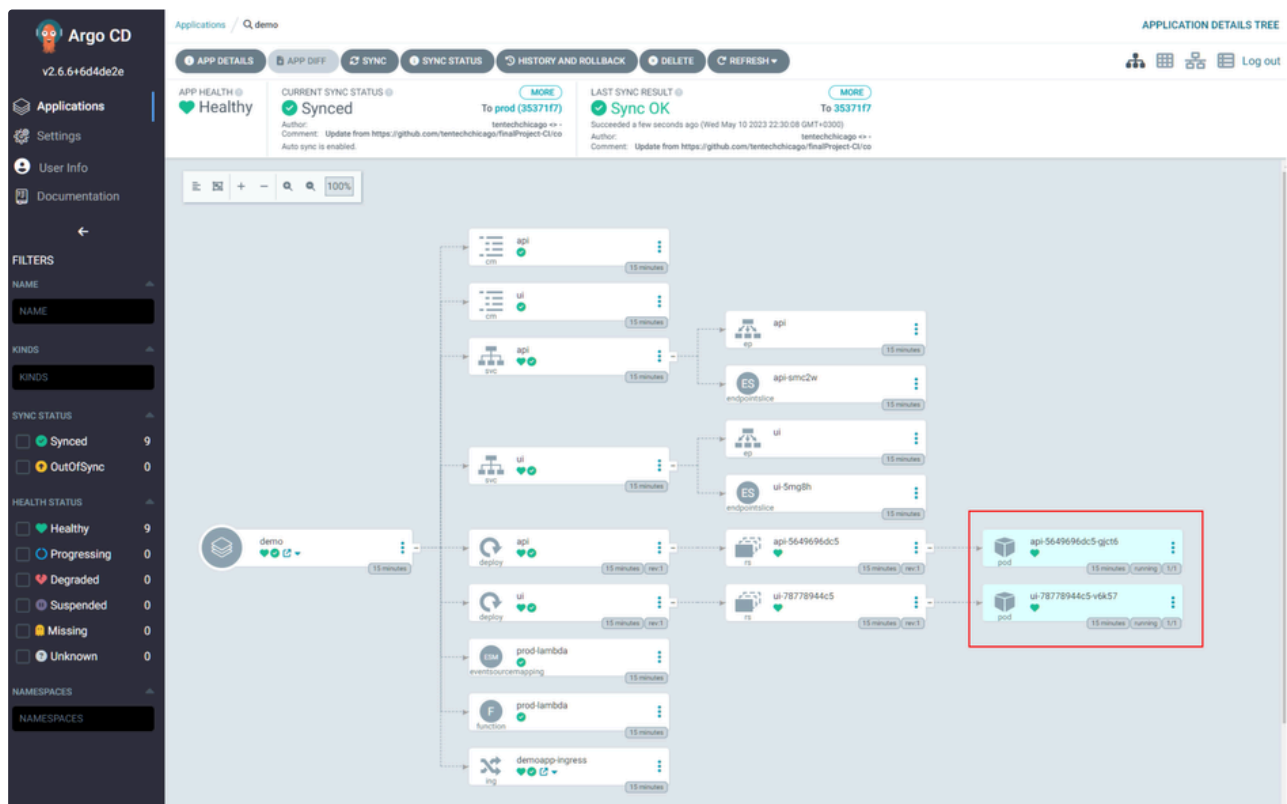
- 1 aws eks update-kubeconfig --name eks-prod
- 2 kubectl -n argocd get secrets argocd-initial-admin-secret -o json | jq -r .data.password | base64 -d

5. Login to argo CD and check sync

<https://argo.prod.your-domain.com/>

login: admin

pass: <your_retrieved_password>



Check the app

<https://demoapp.prod.your-domain.com/>

