

Создание пайплайна в Jenkins

--

Для начала работ вам потребуются следующие инструменты:

- *docker*
- *k3d*
- *gitlab*
- *jenkins*

Подробная установка всех инструментов рассмотрена на предыдущих уроках

--

В веб-интерфейсе jenkins обновим плагин 'Kubernetes':

<input type="checkbox"/>	JavaMail API 1.6.2-9 Library plugins (for use by other plugins) This plugin provides the JavaMail API for other plugins.	14 часов ago	1.6.2-5
<input checked="" type="checkbox"/>	Kubernetes 3883.v4d70a_a_a_df034 Cloud Providers Cluster Management kubernetes Agent Management This plugin integrates Jenkins with Kubernetes	2 часа 33 минут ago	3734.v562b_b_a_627ea_c
<input type="checkbox"/>	SSH server 3.275.v9e17c10f2571 Adds SSH server functionality to Jenkins, exposing CLI commands through it.	2 месяца 5 дня ago	3.236.ved5e1b_cb_50b_2

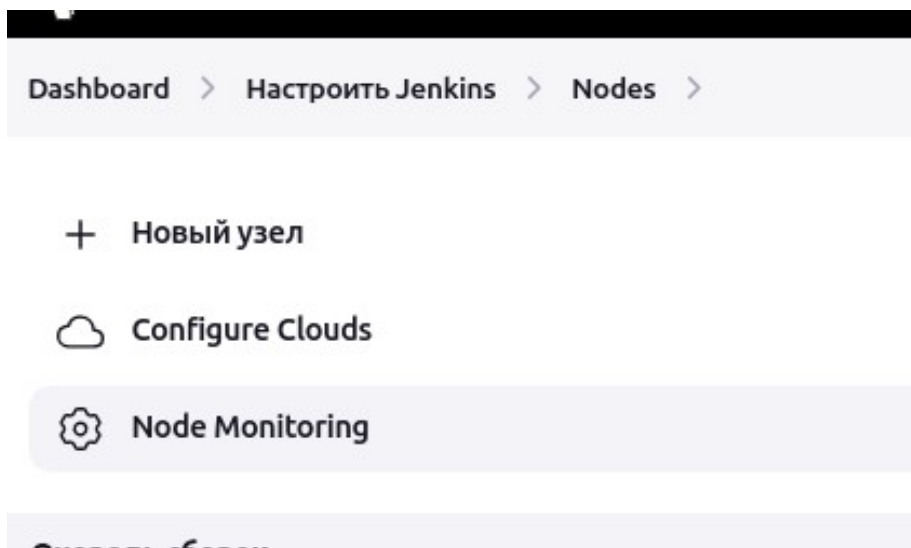
Загрузить и установить после перезагрузки

Обновлено: 5 минуты 17 секунд назад

Проверить сейчас

После обновления потребуется перезагрузка

Переходим в раздел "Управление средами сборки" в настройках Jenkins:



Выбираем пункт "Configure Clouds", убеждаемся, что в поле "Kubernetes Namespace" выбран корректный namespace:

Kubernetes Namespace

jenkins

JNLP Docker Registry ?

Создаем проект с типом "Pipeline" и называем его "jenkins-project-1".
В разделе Pipeline - Script добавляем скрипт пайплайна (запуск shell в kubernetes pod):

```
podTemplate(containers: [  
  containerTemplate(  
    name: 'jnlp',  
    image: 'jenkins/inbound-agent:latest'  
  )  
) {  
  
  node(POD_LABEL) {  
    stage('Get a Maven project') {  
      container('jnlp') {  
        stage('Shell Execution') {  
          sh '''  
            echo "Hello! I am executing shell"  
          '''  
        }  
      }  
    }  
  }  
}
```

```
    }  
  }  
}
```

Сохраняем изменения в проекте и нажимаем в главном меню проекта "Собрать сейчас"

Во время работы пайплайна, переходим в терминал и выполняем команду просмотра pod в неймспейсе "jenkins":

```
$ kubectl get pods -n jenkins
```

```
filipp@filipp-notebook:~$ kubectl get pods -n jenkins  
NAME                                READY   STATUS    RESTARTS   AGE  
jenkins-0                           2/2     Running   1           27m  
jenkins-project-1-3-nqdb7-sljkb-233h5 1/1     Running   0           4s
```

Убеждаемся, что сборка успешно завершена:



Сборка успешно выполнена, переходим в консольный вывод сборки:

```
Running on jenkins-project-1-3-nqdb7-sljkb-233h5 in /home/jenkins/agent/workspace/jenkins-project-1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Get a Maven project)
[Pipeline] container
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Shell Execution)
[Pipeline] sh
+ echo Hello! I am executing shell
Hello! I am executing shell
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // container
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] }
[Pipeline] // podTemplate
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Консольный вывод говорит о том, что сборка успешно выполнена

--

Создадим в gitlab проект под названием "jenkins-shell-project" (сохраните URL проекта, например из адресной строки)"

New project > Create blank project

Project name

jenkins-shell-project

Project URL

http://172.17.0.3:8929/

gitlab-instance-6435c815

Project slug

jenkins-shell-project

Want to organize several dependent projects under the same namespace? [Create a group](#).

Visibility Level ?

- ☒ Private
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.
- ☐ Internal
The project can be accessed by any logged in user except external users.
- ☐ Public
The project can be accessed without any authentication.

Project Configuration

- ☐ Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.
- ☐ Enable Static Application Security Testing (SAST)
Analyze your source code for known security vulnerabilities. [Learn more](#).

Create project

Cancel

Переходим в терминал и выполняем создание локального проекта:

```
$ git init --initial-branch=main
```

```
filipp@filipp-notebook:~/Desktop/jenkins-shell-project$ git init --initial-branch=main
Initialized empty Git repository in /home/filipp/Desktop/jenkins-shell-project/.git/
```

Привяжем удаленный репозиторий к локальному:

```
$ git remote add origin http://172.17.0.2:8929/gitlab-instance-6435c815/jenkins-shell-project.git
```

```
filipp@filipp-notebook:~/Desktop/jenkins-shell-project$ git remote add origin http://172.17.0.2:8929/gitlab-instance-6435c815/jenkins-shell-project.git
```

Создадим файл 'Jenkinsfile'

```
$ nano Jenkinsfile
```

```
filipp@filipp-notebook:~/Desktop/jenkins-shell-project$ nano Jenkinsfile
```

Наполним файл содержимым нашего пайплайна:

```
podTemplate(containers: [  
    containerTemplate(  
        name: 'jenkins-agent',  
        image: 'jenkins/jenkins-agent:latest',  
        command: ['sh'],  
        args: []  
    )  
])
```

```

        name: 'jnlp',
        image: 'jenkins/inbound-agent:latest'
    )
  }) {

    node(POD_LABEL) {
      stage('Get a Maven project') {
        container('jnlp') {
          stage('Shell Execution') {
            sh '''
              echo "Hello! I am executing shell"
            '''
          }
        }
      }
    }
  }
}

```

Сохраним изменения, проиндексируем их, выполним создание коммита и отправим их в удаленный репозиторий:

```

$ git add .
$ git commit -m 'Jenkinsfile added'
$ git push -u origin main

```

```

filipp@filipp-notebook:~/Desktop/jenkins-shell-project$ git add .
filipp@filipp-notebook:~/Desktop/jenkins-shell-project$ git commit -m 'Jenkinsfile added'
[main (root-commit) 64bb7bc] Jenkinsfile added
 1 file changed, 20 insertions(+)
 create mode 100644 Jenkinsfile
filipp@filipp-notebook:~/Desktop/jenkins-shell-project$ git push -u origin main
Username for 'http://172.17.0.3:8929': root
Password for 'http://root@172.17.0.3:8929':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 413 bytes | 413.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To http://172.17.0.3:8929/gitlab-instance-6435c815/jenkins-shell-project.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

```

Для того, чтобы Jenkins (оркестрируемый k3d) смог подключиться к Gitlab, необходимо добавить контейнеры в одну docker-сеть:

```

$ docker network connect bridge k3d-mycluster-serverlb
$ docker network connect bridge k3d-mycluster-server-0

```

```

filipp@filipp-notebook:~$ docker network connect bridge k3d-mycluster-serverlb
filipp@filipp-notebook:~$ docker network connect bridge k3d-mycluster-server-0

```

Проверим, что все контейнеры (и k3d и Gitlab) находятся в одной docker-сети:


```
$ docker network inspect bridge
```

```
"Containers": {
  "3046d13b3629344b459252bba985f50649319995dafce242c4a4df7461d532ec": {
    "Name": "k3d-mycluster-server-0",
    "EndpointID": "a9bc523e504ee6877c55007249aae75cbc728def42c3dd139c37149
6943c23a6",
    "MacAddress": "02:42:ac:11:00:04",
    "IPv4Address": "172.17.0.4/16",
    "IPv6Address": ""
  },
  "3b9937e4ead0d1f13fb7f649b05f08e3349d7e77a0b1cac2991f12375b535c10": {
    "Name": "gitlab_web_1",
    "EndpointID": "34682c98283c4b18f23023b8450f8fea8d2e7366a03f691011f42d6
f4b6bfb13",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "IPv6Address": ""
  },
  "cf665af888f478c215096c95cd70ad9033e1d146485fa29d62e41b80e7ecd3e2": {
    "Name": "k3d-mycluster-serverlb",
    "EndpointID": "e5a9f9da9b5c997fed3c0eabb08f8dcc0c712d7cb21ccf9a181869d
e36857eba",
    "MacAddress": "02:42:ac:11:00:02",
    "IPv4Address": "172.17.0.2/16",
    "IPv6Address": ""
  }
}
```


Перейдем в Jenkins и создадим новый проект с типом "Pipeline":

Введите имя Item'a


» Обязательное поле

**Создать задачу со свободной конфигурацией**

Это - основной и наиболее универсальный тип задач в Jenkins. Jenkins будет собирать ваш проект, комбинируя любую SCM с любой сборочной системой, отличных от сборки ПО.

**Pipeline**

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex builds.

**Folder**

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace for the items they are in different folders.

В настройках преокта, в разделе Pipeline, в качестве значения выпадающего списка "Definition" выбираем "Pipeline from SCM".

Далее: "SCM - Git", "Repository URL - URL нашего проекта в gitlab, где хранится Jenkinsfile. Добавляем новые credentials с помощью кнопки "Add" (все настройки в открывшемся окне оставляем по умолчанию и заполняем логин и пароль от используемого пользователя gitlab):

Domain

Global credentials (unrestricted) ▼

Kind

Username with password ▼

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?

root

☐ Treat username as secret ?

Password ?

.....

После сохранения учетных данных, указываем их в выпадающем списке, а в настройке "указателе на ветку" выбираем ветку main:

Git

?

Repositories ?

Repository URL ?

✕

http://172.17.0.3:8929/gitlab-instance-6435c815/jenkins-shell-project

Credentials ?

root/*****

▼

+ Add

Расширенные...

Add Repository

Branches to build ?



Branch Specifier (blank for 'any') ?


✕


*/main

Add Branch


Сохраняем все настройки, переходим в основное меню проекта Jenkins и нажимаем кнопку "Собрать сейчас".


 **История сборок** **тренд** 



 **#1**

| **17 февр. 2023 г., 15:52**

 **Atom feed для всех**

 **Atom feed для неудачных**

Сборка завершена, переходим в консольный вывод сборки:

```
Running on jenkins-project-2-1-c0hr7-t0hv5-6lhwx in /home/jenkins/agent/workspace/jenkins-project-2
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Get a Maven project)
[Pipeline] container
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Shell Execution)
[Pipeline] sh
+ echo Hello! I am executing shell
Hello! I am executing shell
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // container
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] }
[Pipeline] // podTemplate
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Мы создали два проекта в Jenkins - с пайплайном внутри проекта, и в виде файла Jenkinsfile, хранящегося в gitlab. Выполнили интеграцию с gitlab из jenkins, сконфигурировали запуск рабочих сборочных нод в kubernetes pod в неймспейсе jenkins.

--