Doktor Plugin

Plugin Information
View Doktor on the plugin site for more information.

Jenkins plugin for automated documentation uploading to Confluence.

Usage

Features

Supported formats

Doktor supports Markdown and AsciiDoc. Markdown support is provided by awesome flexmark-java library. AsciiDoc is supported thanks to AsciidoctorJ. Please note that AsciiDoc support is very experimental.

Front matter

Front matter is another word for *metadata*, used by bloggers and hipsters widely. Doktor uses front matter to configure the way pages appear in Confluence.

Markdown

Doktor supports YAML front matter in your Markdown files. Front matter looks like a small YAML fragment at the beginning of the file, separated by a triple minus sign (`---`) in this case:

```
---
key: value
---
```

AsciiDoc

Doktor supports YAML front matter in your AsciiDoc files as well. Front matter looks like a small YAML fragment at the beginning of the file, separated by a triple minus sign (`---`) in this case. Note, that due to a more strict YAML parser logic for AsciiDoc, strings with special characters need to be quoted:

```
---
key: value
key_with_specials: 'value: with specials'
---
```

Supported front matter attributes:

titleRequired. Page title, unsurprisingly.parentOptional. Parent page title, if any. If omitted, "orphaned" page will be created. If parent page is not found by title, child page will not be created at all.labelsOptional. List of labels to add to a page.

Tables

Markdown

Though Markdown does not have any support for tables, Doktor supports GitHub Flavored Markdown tables. You can also create tables by inlining XHTML markup directly in your docs.

AsciiDoc

AsciiDoc (thus Asciidoctor and AsciidoctorJ) supports tables natively.

Images

Doktor supports images. When an image is referred by relative URL it will be uploaded to a Confluence server as an attachment of a page, given unique name. When an image is referred by remote URL (Internet link) it will be referred by this URL from a Confluence server.

Markdown

Markdown syntax for images:

```
![Millennium Falcon](./millennium_falcon.png "The Millennium Falcon, Han Solo's most prized possession")
```

AsciiDoc

Images look like this is AsciiDoc:

```
.The Millennium Falcon, Han Solo's most prized possession
[link=http://starwars.wikia.com/wiki/Millennium_Falcon]
image::./millennium_falcon.png[Millennium Falcon,400,float="right",align="center"]
```

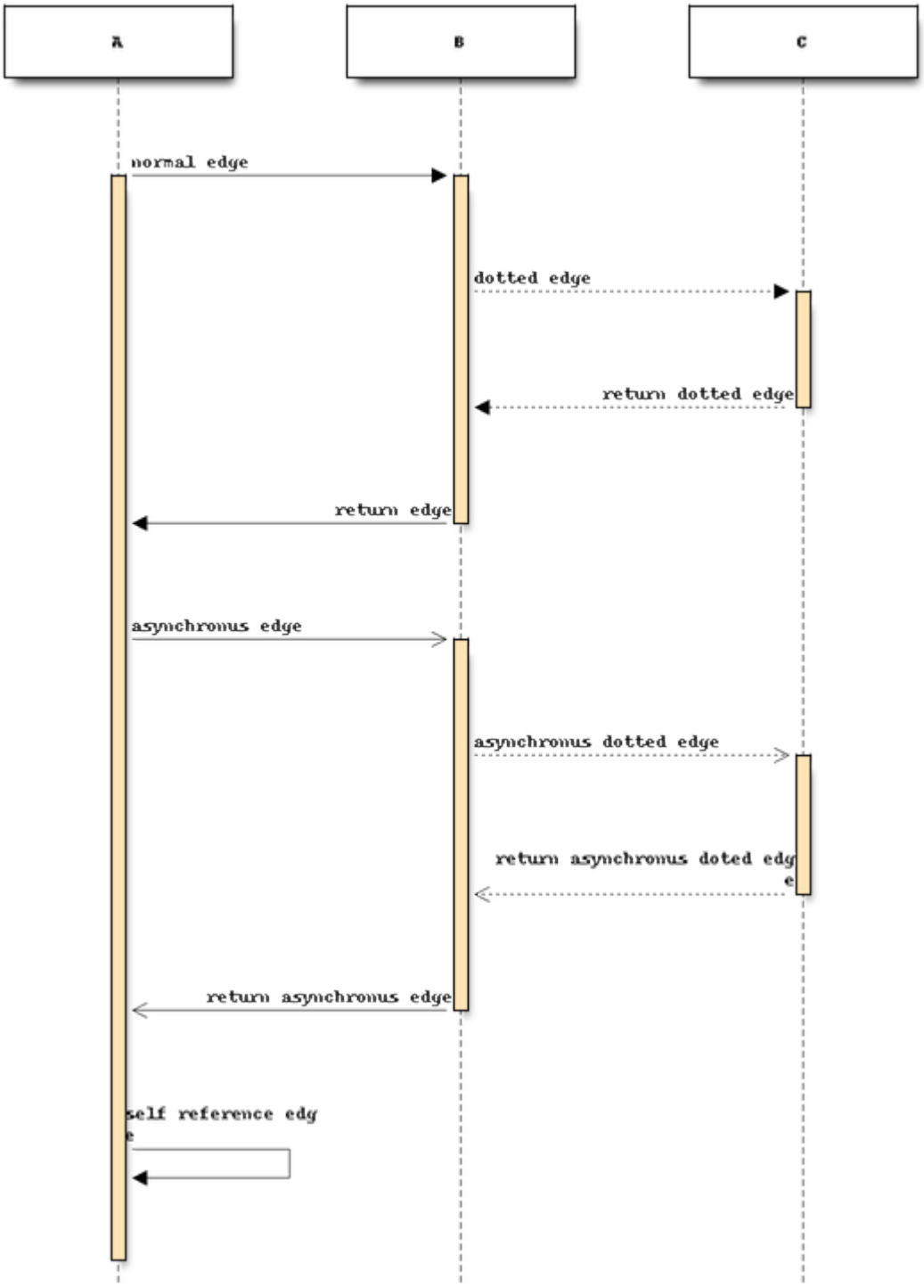
As you see, AsciiDoc is more feature-rich.

Diagrams

Diagrams are only supported in AsciiDoc, the markup looks like this:

```
seqdiag {
  // normal edge and doted edge
  A -> B [label = "normal edge"];
  B --> C [label = "dotted edge"];

  B <-- C [label = "return dotted edge"];
  A <- B [label = "return edge"];

  // asynchronus edge
  A ->> B [label = "asynchronus edge"];
  B -->> C [label = "asynchronus dotted edge"];

  B <<-- C [label = "return asynchronus doted edge"];
  A <<- B [label = "return asynchronus edge"];

  // self referenced edge
  A -> A [label = "self reference edge"];
}
```

The snippet above will be rendered in this image:

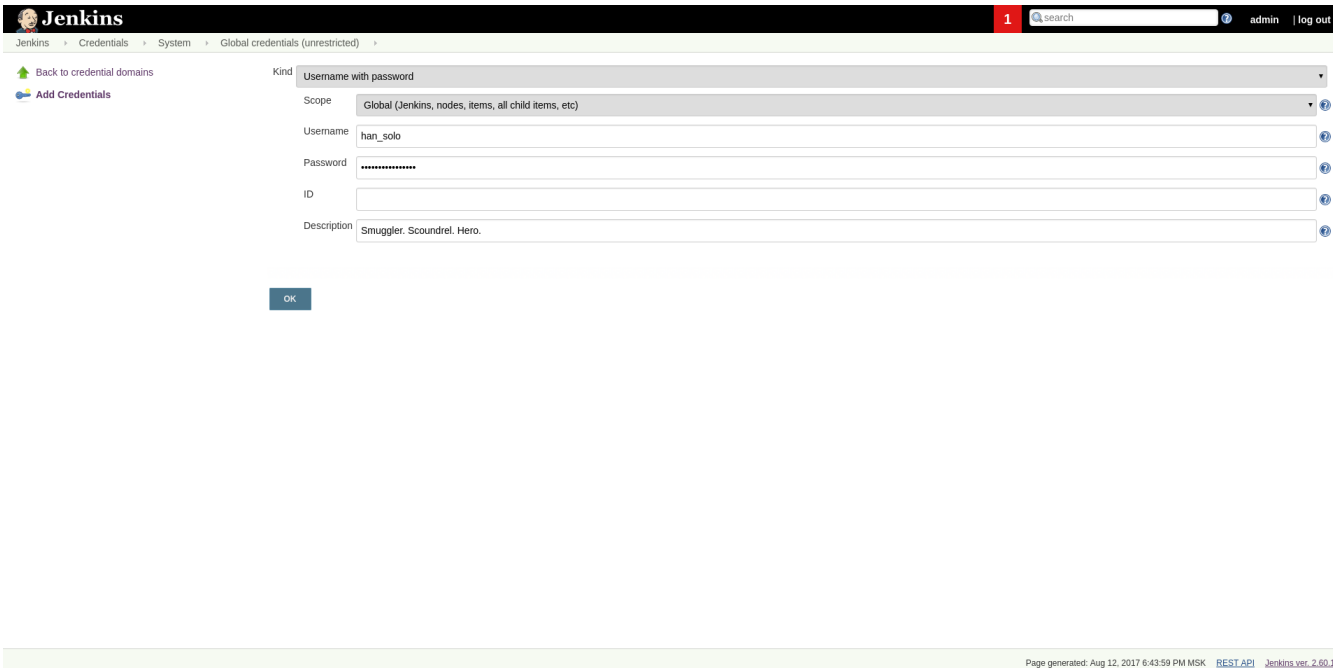


Read more about diagram syntax in in the official AsciiDoc guide. Be warned, that most types of diagrams require external tools (like `seqdia` or `dot`) to be installed and available on the `PATH`.
Currently, these diagrams are supported:

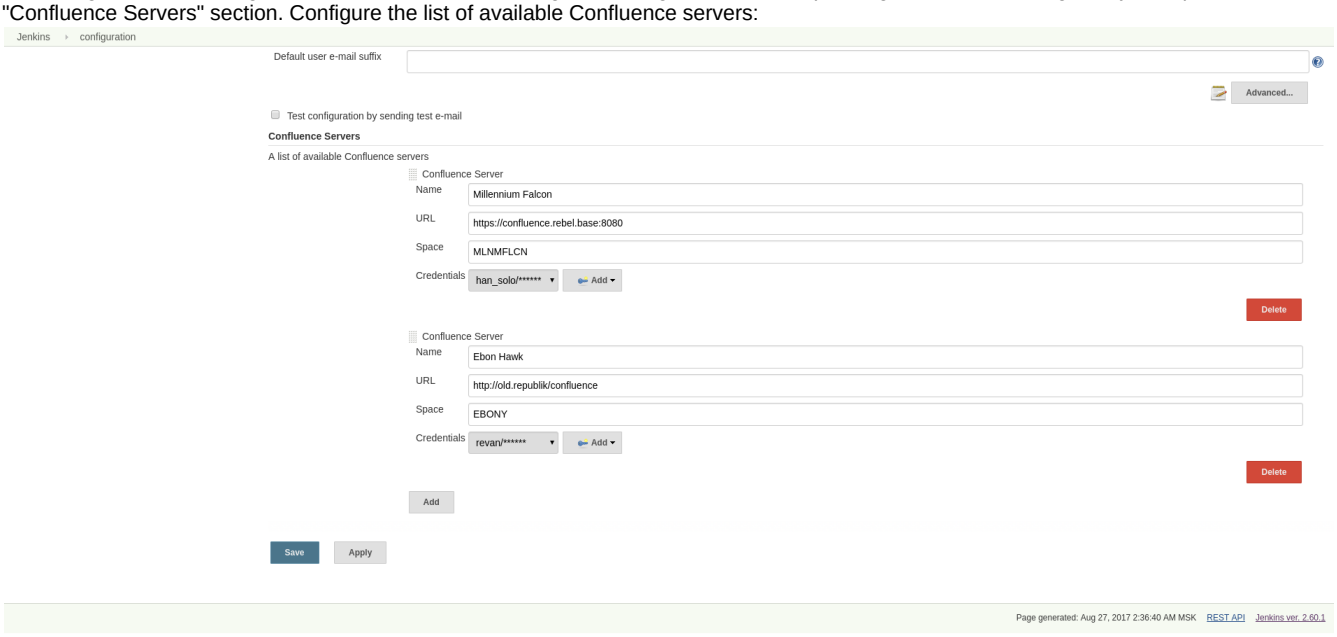
- `actdiag` / `blockdiag` / `nwdiag` / `packetdiag` / `rackdiag` / `seqdiag`. These diagrams require `blockdiag` and related Python packages to be available on the `PATH`.
- `ditaa`. No additional tools needed.
- `graphviz`. Obviously, requires Graphviz tool to be on the `PATH`.
- `mermaid` Requires mermaid (version prior to `7.x`) and PhantomJS to be on the `PATH`.
- `plantuml` No additional tools needed.

Configure Confluence servers

As you might suspect, Confluence REST API requires authentication. Doktor supports basic authentication (username and password). So, first thing to do is to configure credentials in Jenkins.
Create a "Username with password" credentials to be used to authenticate on Confluence server:



You may have as many Confluence servers and credentials for them as you need.
Next thing to do is to configure Confluence servers. Go to global configuration screen ("Manage Jenkins" "Configure System") and find "Confluence Servers" section. Configure the list of available Confluence servers:



Now, when you have some Confluence servers to publish documentation to, it's time test this plugin! Yes, I'm using word "test" intentionally here.

Pipeline step

Using Doktor with pipelines is very easy! Here is the full syntax of `doktor` step:

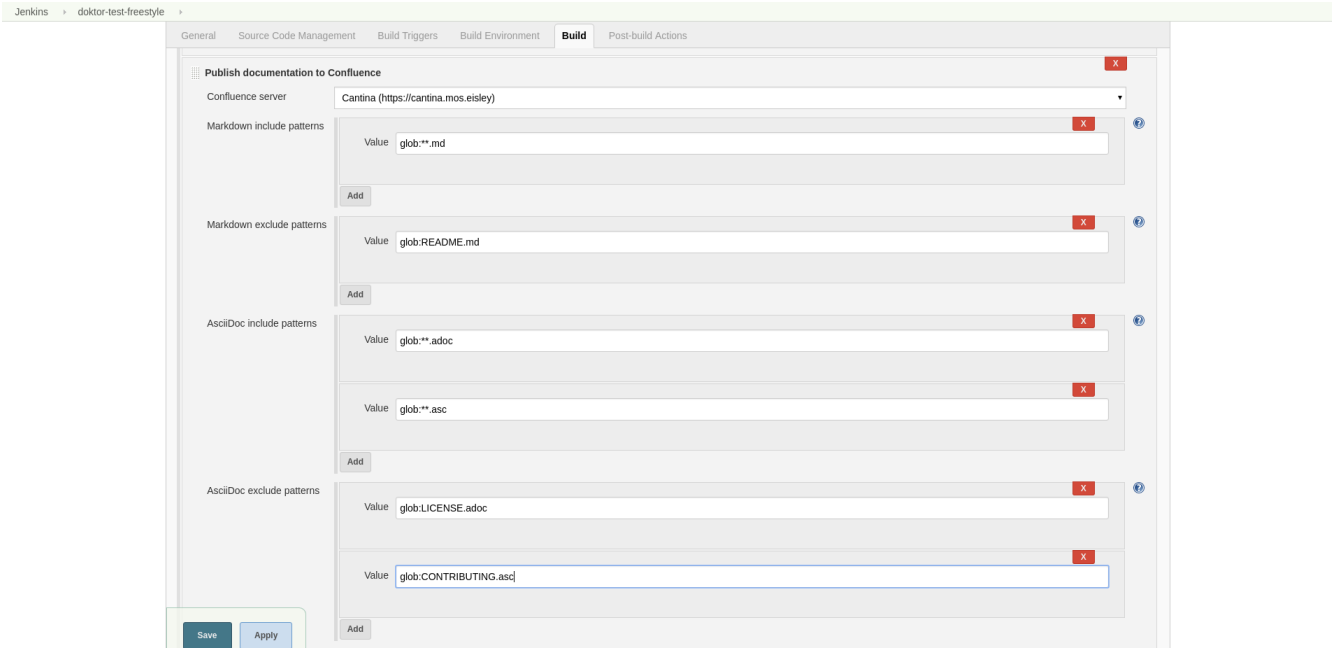
```
doktor
        server : 'Cantina', (1)
        markdownIncludePatterns: ['glob:**.md'], (2)
        markdownExcludePatterns: ['glob:README.md'], (3)
        asciidocIncludePatterns: ['glob:**.adoc', 'glob:**.asc'], (4)
        asciidocExcludePatterns: ['glob:LICENSE.adoc', 'glob:CONTRIBUTING.asc'] (5)
```

1. One of the available Confluence servers
2. List of Java 8 PathMatcher specifications for Markdown files to include.
3. List of Java 8 PathMatcher specifications for Markdown files to exclude.
4. List of Java 8 PathMatcher specifications for AsciiDoc files to include.
5. List of Java 8 PathMatcher specifications for AsciiDoc files to exclude.

You can also try your luck with "Snippet Generator", available at `/pipeline-syntax` path of your Jenkins installation.

Classic builds

Doktor plays nice with "classic" builds too!



Click those question icons on the right if you need any help.

Limitations

Doktor recreates pages instead of updating them. Recreating pages has some counterintuitive effects:

- Any modification will overridden on each Doktor run, obviously
- Page likes are not preserved
- Attachments are not preserved
- There is no support for extra Confluence markup, macroses and features like comments

This may sound shocking to you, but let me explain.
Doktor's idea is just uploading your documentation somewhere, making it available to *read* by everybody. Doktor is not about collaborative editing - use VCS for that. It's a unidirectional flow - from sources to rendered documents - by design. I was inspired by GitHub's pages and wikis, and I sincerely believe in this approach.
At the moment, Doktor supports only Confluence and may never support any other services (unless my employer switches to another vendor).

Developing

Doktor is built with Kotlin, Gradle and Love. Well, actually with hate to the workflows on my day-time job.
JPI artifact is produced with Gradle's JPI plugin. Read its documentation to know more about supported features and options.
Also, take a look at this awesome Jenkins plugin, which is build with Gradle and Kotlin too!

Building & running

Basically, `./gradlew --rerun-tasks clean jpi server` will spin up a Jenkins with Doktor installed. `--rerun-tasks` is used to force clean build every time because Gradle aggressively caches build outputs, especially Kotlin annotation processing tool results. Feel free to tweak CLI arguments, assuming you know what you do.
Debug is supported as well:

```
GRADLE_OPTS="-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005" ./gradlew --rerun-tasks clean jpi server
```

Omit `server` task if you just need a JPI file.

Testing on remote agents

Once you may want to test how Doktor behaves on agents. The simplest way to do that is to run an agent in Docker. There are two images for agents available.

jenkinsci/slave

jenkinsci/slave is an image meant to be run by Jenkins to start a new agent. The configuration is very simple:

When you're running Jenkins via Gradle JPI plugin it will be run under you user account, so either your user needs to be able to execute `sudo docker` without password or you will need to type that password in Gradle's terminal session.

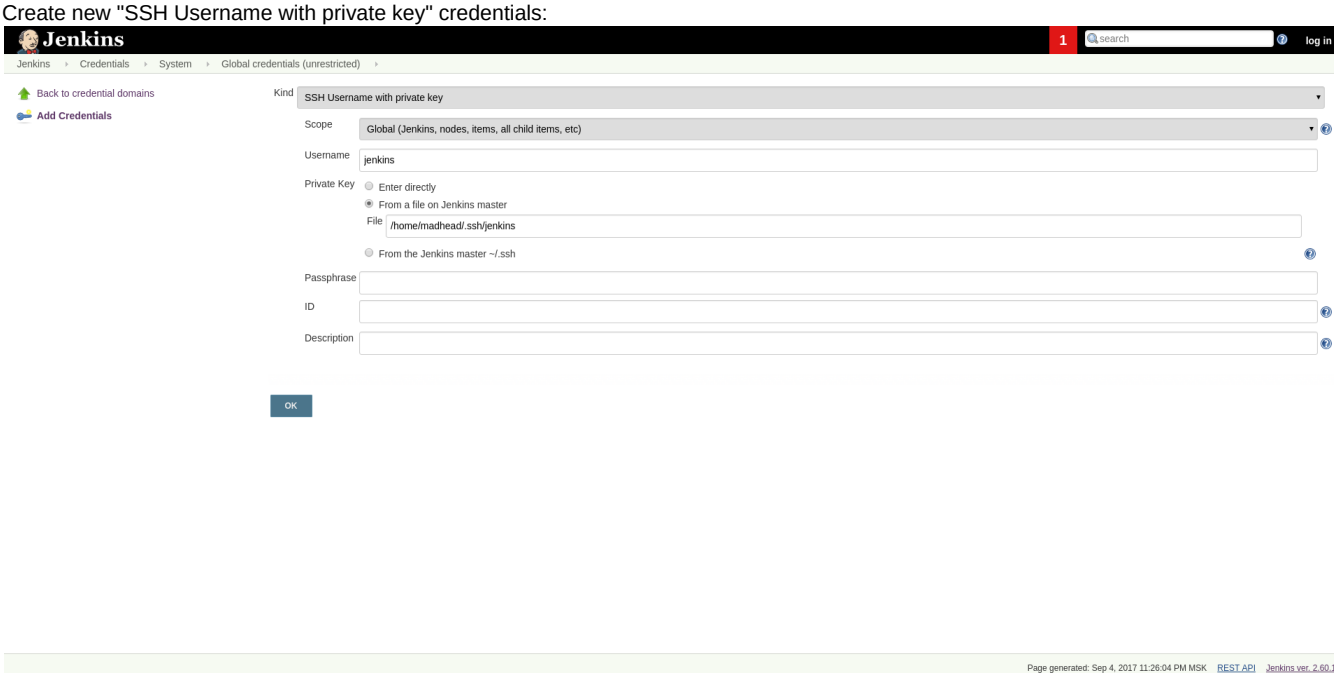
jenkinsci/ssh-slave

jenkinsci/ssh-slave is another (better) option. It allows you manage agent container separately and then attach it to Jenkins, thus eliminating the need to provide any password or execute `sudo docker`. Container's mounts and FS modifications will be preserved between Jenkins restarts.

First, you need to have an SSH key pair that will be used to connect to the agent. Looks like only RSA keys are supported (public key must start with `ssh-` prefix). Either create a new one, or use the existing.

Then, install SSH Slaves plugin on the master.

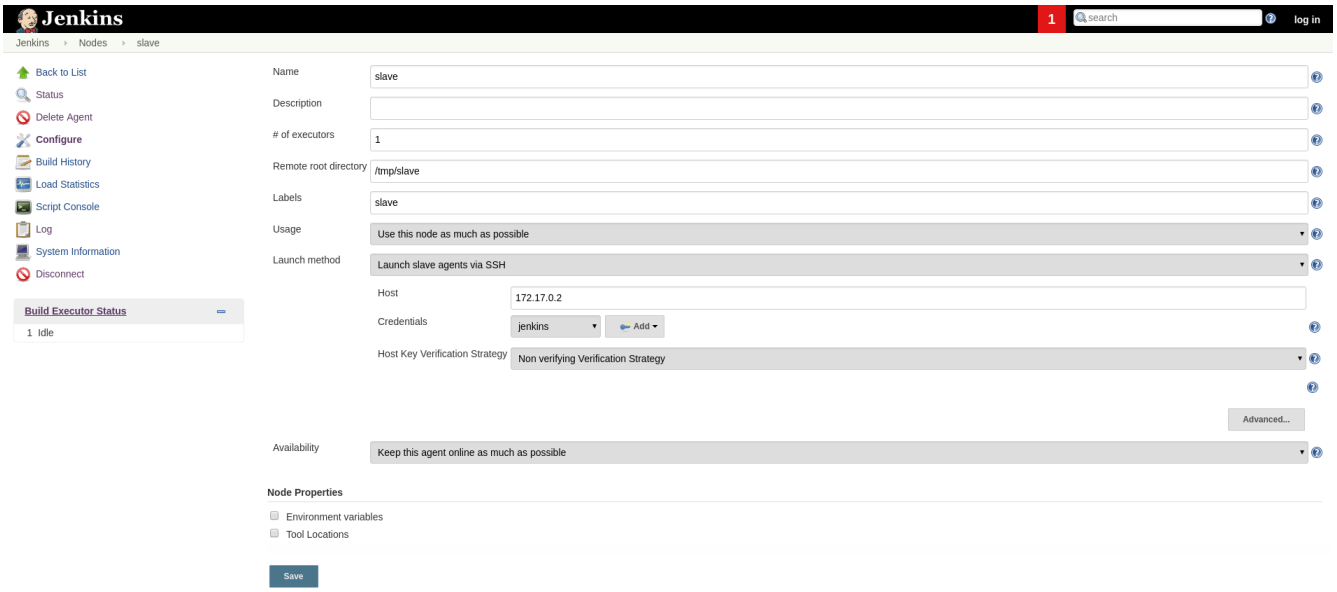
Create new "SSH Username with private key" credentials:



You can paste private key directly here or use one of the defaults (`~/.ssh/id_ecdsa`, `~/.ssh/id_rsa`, `~/.ssh/id_dsa`, `~/.ssh/identity`).

Next, start agent container by executing `docker run --detach --name jenkins-slave jenkinsci/ssh-slave "$(cat ~/.ssh/jenkins.pub)"` (assuming that `~/.ssh/jenkins.pub` is a public key corresponding to the private key from previous step).

Finally, create new agent with a configuration like this:



`172.17.0.2` here is the IP of a Docker contaner from the previous step, found in `docker inspect` output. You could also run the container exposing the ports (e.g. `-p 2222:22`) and then use `localhost` as host and `2222` as port.

Testing Confluence integration

You'll need to refer to Confluence REST API. Here is the link. Samples are also available.

Cloud

Probably, the easiest (and CPU / RAM saving) way to run Confluence is to run it in the cloud (AWS EC2, DigitalOcean, …). Though, it will cost you some money.
There is an Ansible script in this repo to automate Confluence installation. It assumes that you already have a running instance that meets Confluence's minimal system requirements. Read your cloud provider's documentation to know how to create and manageVMs.
When you have a VM, just follow these steps to install Confluence Server:

1. Create inventory file (`.ansible/inventory`) with a content like this:

   ```
   [confluence]
   your.confluence.host
   ```

   You might want to add additional parameters. For example, a set of parameters for Ubuntu 16.04 EC2 instance:
   ```
   [confluence]
   your.confluence.host ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/confluence.pem ansible_python_interpreter=/usr/bin/python3
   ```

   Or you can just use dynamic inventories.
2. Install required roles from Ansible Galaxy: `sudo ansible-galaxy install -r requirements.yml --force`.
3. After the inventory is configured, just run `./confluence.yml` from the `.ansible` directory.
4. Go to `http://your.confluence.host/` (if the DNS and IPs are set) and configure the instance. Note, that you will need a license key (trial works for 90 days).

Docker

You can run Confluence locally as well. The easiest way here is Docker (Windows uses should appreciate the joke).
Running Confluence is as simple as:

```
docker volume create --name confluence-data
docker run --detach --volume confluence-data:/var/atlassian/application-data/confluence --name confluence --publish-all atlassian/confluence-server:latest
```

You might want to add some additional options or tweak the existing ones.
Note, that you will need a license key (trial works for 90 days).