Use Docker images as build environments

Bitbucket Pipelines runs your builds in Docker containers. These containers run a Docker image that defines the build environment. You can use the default image provided by Bitbucket or get a custom one.

We support public and private Docker images including those hosted on Docker Hub, AWS, GCP, Azure and self-hosted registries accessible on the internet. (Bitbucket Pipelines cannot currently access Docker images that cannot be accessed via the internet.)

Overview

Default build environment

If you don't specify a Docker image to use as your build environment, Bitbucket Pipelines will use a default one that we have built with some common tools.

The default image is `atlassian/default-image:latest`. You can also specify the version:

Name	Platform	Applications available out-of-the-box
atlassian/default-image:1 atlassian/default-image:latest	Ubuntu 14.04	• wget • xvfb • curl • ssh • git: 1.9.1 • mercurial: 2.8.2 • java: 1.8u66 • maven: 3.0.5 • node: 4.2.1 • npm: 2.14.7 • nvm: 0.29.0 • python: 2.7.6 • gcc: 4.8.4 • ant: 1.9.3
atlassian/default-image:2	Ubuntu 16.04	• wget • xvfb • curl • ssh • git: 2.7.4 • mercurial: 3.7.3 • java: Open-JDK 1.8u151 • maven: 3.3.9 • node: 8.9.4 • npm: 5.6.0 • nvm: 0.33.8 • python: 2.7.12 • gcc: 5.4.0 • ant: 1.9.6
atlassian/default-image:3	ubuntu 20.04 LTS	• wget • xvfb • curl • ssh • zip • lfs • tar • parallel • git: 2.25.1 • node: 14.17.0 • npm: 6.14.14 • nvm: 0.39.2 • python: 3.8.10 • gcc: 9.3.0 • ant: 1.10.7
atlassian/default-image:4 **Recommended**	ubuntu 22.04 LTS	• wget • xvfb • curl • ssh • zip • lfs • tar • parallel • git: 2.39.1 • node: 18.13.0 • npm: 8.19.3 • nvm: 0.39.2 • python: 3.10.6 • gcc: 11.3.0 • ant: 1.10.12

Best practices tip: after you have Pipelines working, try to find a specific image you can use to not rely on our default image. We don't upgrade it very often, to maximize compatibility for customers who rely on it, and it includes many tools you might not need. This means you'll likely get bug fixes and faster builds from using your own, smaller build image.

Custom build environments

If the default environment does not exactly meet your needs, your first step should be to see if a public available image on Docker Hub will help. Here are some of the official images available on Docker Hub that we recommend and configure in Pipelines during setup:

- PHP: https://hub.docker.com/_/php/
- JavaScript: https://hub.docker.com/_/node/
- Java (Maven): https://hub.docker.com/_/maven/
- Python: https://hub.docker.com/_/python/
- Go: https://hub.docker.com/_/golang/
- .NET Core: https://hub.docker.com/r/microsoft/dotnet/

To use these images or another public image on Docker Hub as your build image, see the guide for *Using public build images* below.

To use an existing Docker image from your own registry, see the guide for *Using private build images* below.

To build your own Docker images, see the guide for *Creating a custom build environment* below.

Using public build images

You can use existing public images hosted on Docker Hub, another registry, or in a self-hosted registry. You can only use images that can be accessed via the internet.

In the examples below:

- The account name is the name of the account that owns the image.
- The username, password, and email are your personal credentials for the registry.

Public images hosted on Docker Hub

You can find the name of the image by looking at the `pull` command listed (1) on the relevant dockerhub page.

If you don't specify the tag, Docker uses the `latest` version tag:

```
image: openjdk
```

Here's an example with both image version and account on Docker Hub:

```
image: account-name/openjdk:8
```

Public images hosted outside Docker Hub

If the image is not provided by Docker and is hosted in a private registry, the image name should include the URL:

```
image: docker.someprovider.com/account-name/openjdk:8
```

Using private build images

To authenticate with a private Docker registry, including self-hosted registries and private images on Docker Hub, Amazon ECR and Google GCR, you need to provide a username and password as part of the image configuration in your YAML file. You can also optionally include an email address, which may be required by some providers.

Private images hosted by Docker Hub

You can use secure variables to configure username and password variables, then add them to the image YAML configuration as shown below:

```
image:
  name: account-name/openjdk:8
  username: $DOCKER_HUB_USERNAME
  password: $DOCKER_HUB_PASSWORD
  email: $DOCKER_HUB_EMAIL
```

Private images hosted by AWS ECR (EC2 Container Registry)

If the image is hosted by ECR, you can provide the access key and secret key via secure variables under an additional "aws" section in your YAML image configuration:

```
image:
  name: <aws_account_id>.dkr.ecr.<region>.amazonaws.com/openjdk:8
  aws:
    access-key: $AWS_ACCESS_KEY
    secret-key: $AWS_SECRET_KEY
```

Alternatively, you can avoid storing AWS_ACCESS_KEY and AWS_SECRET_KEY in Bitbucket and make use of OpenID Connect functionality to allow your workspace builds to access your image only.

```
image:
  name: <aws_account_id>.dkr.ecr.<region>.amazonaws.com/openjdk:8
  aws:
    oidc-role: arn:aws:iam::<aws_account_id>:role/<your_role_name>
```

For step-by-step instruction on how to configure your AWS Account to work with Bitbucket OpenID Connect, check out the following: Use AWS ECR images in Pipelines with OpenID Connect.

Private images hosted by Google Container Registry (GCR)

To pull an image from GCR, you need to configure a service account for Pipelines with "Viewer" access in your GCP admin console.

Download the generated JSON private key, and copy/paste it into a secure variable in Pipelines. The configured variable can then be used directly in the password field in your YAML image configuration as shown below:

```
image:
  name: <region>.gcr.io/<project>/image:latest
  username: _json_key
  password: '$GCR_JSON_KEY'
```

Images hosted on other registries

To use another private registry, you must provide the registry URL in the image name, and username/password credentials as shown below:

```
image:
  name: docker_your-company-name.com/account-name/openjdk:8
  username: $USERNAME
  password: $PASSWORD
  email: $EMAIL
```

Pin images by digest

To ensure that an image cannot be modified, you can refer to an image by its hash or digest value. The digest is a cryptographic hash, that changes whenever the content of an image changes. Pinning the digest of an image is the surest way to guarantee that no changes can be introduced.

Use image digests

```
image:
  name: ubuntu@sha256:a0ee7647e24c8494f1cf6b94f1a3cd127f423268293c25d924fbe18fd82db5a4
```

Find an image digest

```
> docker inspect --format='{{.RepoDigests}}' ubuntu

[ubuntu@sha256:a0ee7647e24c8494f1cf6b94f1a3cd127f423268293c25d924fbe18fd82db5a4]
```

Override the default user

An image's default user can be overridden by specifying a user UID as the run-as-user. The specified user UID must be an existing user in the image with a valid home directory.

```
image:
  name: atlassian/default-image:3
  run-as-user: 1000
```

Creating a custom build environment

You can build your own Docker file and use it as your build environment. This way, you can keep your build environment lightweight by only including the tools that you need in your build.

Resources

- **Install Docker locally:** https://docs.docker.com/engine/installation/
- **Build your own Docker image:** https://docs.docker.com/build/building/base-images/
- **Publish your Docker image to Docker Hub:** https://docs.docker.com/get-started/workshop/04_sharing_app/
- **How to use your own registry:** https://www.docker.com/blog/how-to-use-your-own-registry-2/