Packaging Software: an end-to-end pipeline

Tuesday, 17th November 2020

vmware®

Escuela Técnica Superior de
Ingeniería Informática

Welcome

- The spoken language of this talk is Spanish, but all **the slides will be in English**.
- Please **provide proactive feedback** on whether you are understanding what we are exposing.
- These slides will be available in One Drive.
- During the talk, some confidential material will be shown to illustrate real scenarios. This material won't be shared, and photos/screenshots are prohibited.

Packaging Software: an end-to-end pipeline

Agenda

I. About Us

II. Packaging Software in Bitnami: History

III. Software packages

IV. How to build a container image?

V. The containers pipeline

VI. A real example

About us

The lecturers

- Software Engineer
- MS Cybersecurity
- Focused on testing Chart applications
- Active maintainer of a catalog of +130 apps

Jose Antonio Carmona Fombella

jcarmona@vmware.com

- Telecommunication Engineer
- MBA
- Active maintainer of the official Helm Charts repository
- Focused on the Bitnami "enterprise" catalog

Carlos Rodríguez Hernández

carlosrh@vmware.com

Bitnami **is an app store with more than 145 Open Source applications and development environments.**

- **Native installers**
- **Virtual machines**
- **Cloud images**
- **Docker containers and Charts**

More information about us

Some numbers

- More than **1M of monthly deployments**
- More than **1.5 M of downloads per year**
- 35.000 daily visits

Our clients

Microsoft Azure

amazon web services

ORACLE CLOUD PLATFORM

Google Cloud Platform

Additionally

- The project was born in Sevilla.
- Profitable since 2012
- No subsidies
- Acquired by **VMware** in May, 2019

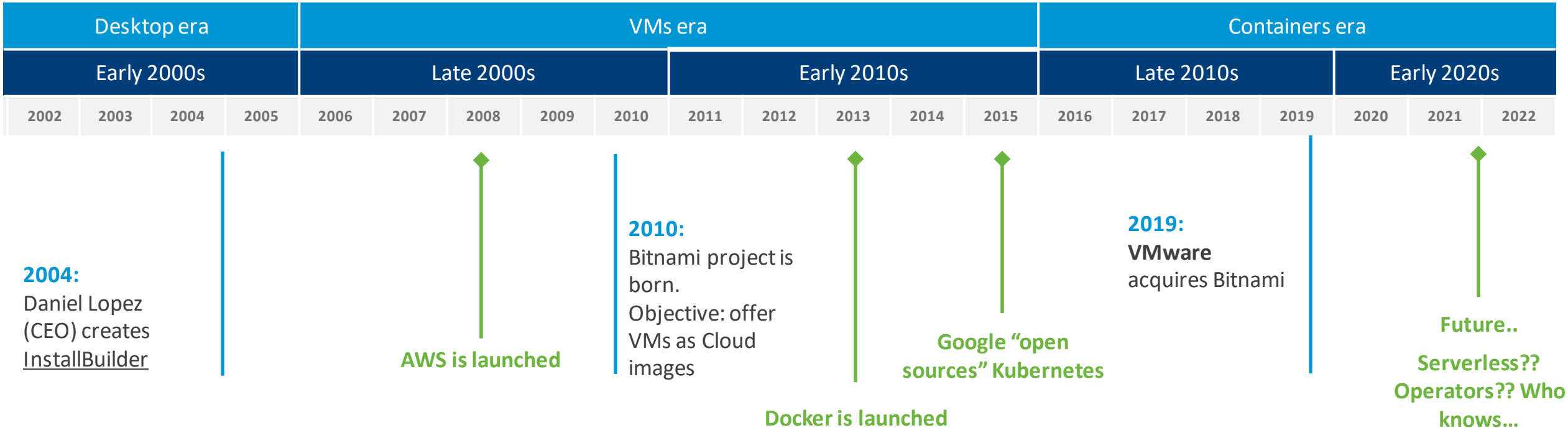
The team

Distributed all over the world, Bitnami used to have a team of ~70 people (30% of the them working remotely) with two offices (Sevilla and San Francisco).

Now, as part of **VMware,** we are part of the Cloud Services team (under the Cloud Services B.U.).

Packaging Software in Bitnami: History

Packaging Software in Bitnami: History



Bitrock offers Applications Native Installers for Windows, OS X and Linux.

Bitrock offers apps as VMs.

Bitnami offers Apps as Cloud Images on every major Cloud Provider

Bitnami offers its application catalog as Docker images and Helm Charts

Software packages

Software packages

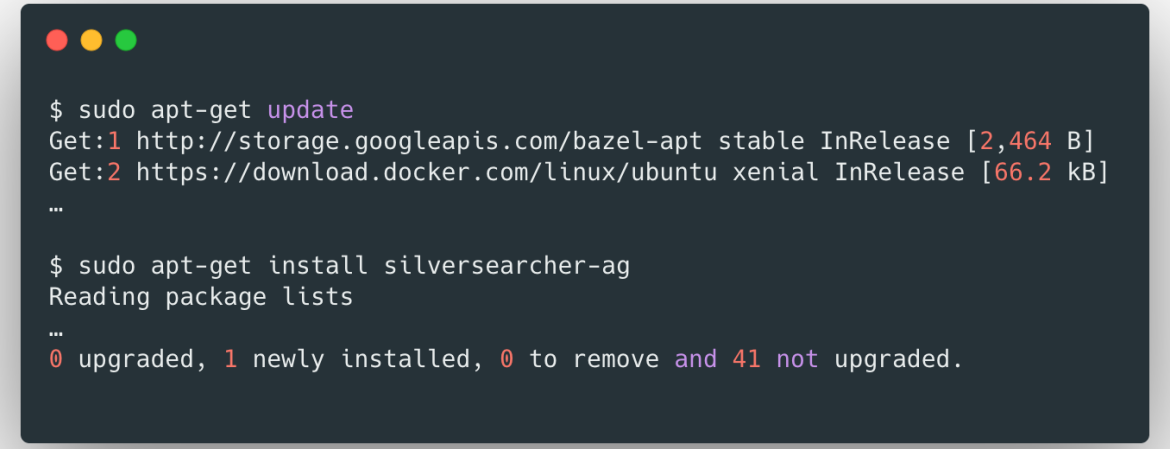
Package Manager Systems

Each Linux Distro has its own PMS which deals with:

- Software repositories and binaries repositories
- Distribution
- Installation
- Updates

Popular PMS:

- yum (CentOS, OracleLinux)
- dkpg/apt (Debian, Ubuntu)
- dnf (Fedora)



According to Wikipedia...

A software package is an archive file containing a computer program as well as necessary metadata for its deployment.

The computer program can be in source code that has to be compiled and built first.

Package metadata include package description, package version, and dependencies (other packages that need to be installed beforehand)

Software packages: what do we use?

Bitnami uses both “system packages” (the ones provided by distros) and “bitnami packages”.

What is a “bitnami package”?

Software packages built by ourselves. One package per “asset”.

To build these packages, it’s necessary to:

- Download the source code (third party).
- Compile to generate the corresponding binaries.
- Add some logic to configure the assets.
- Add some metadata.
- Test them.

All this process is automated in a **pipeline**. (It will be described later)

How to build a container image?

Docker Images


```
FROM bitnami/minideb:stretch
LABEL maintainer "Bitnami <containers@bitnami.com>"

ENV OS_ARCH="amd64" \
    OS_FLAVOUR="debian-9" \
    OS_NAME="linux" \
    ...

# Install required system packages and dependencies
RUN install_packages ca-certificates curl libc6 ...
RUN . ./libcomponent.sh && component_unpack "nginx" "1.16.1-3"
...

COPY rootfs /
RUN /post-unpack.sh
ENV BITNAMI_APP_NAME="nginx" \
    BITNAMI_IMAGE_VERSION="1.16.1-debian-9-r99" \
    PATH="/opt/bitnami/nginx/sbin:$PATH" \
    ...

EXPOSE 8080 8443
WORKDIR /app
USER 1001
ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "/run.sh" ]
```

Most extended solution to create/run containers.

Container images are built reading a series of instructions written on a “Dockerfile”.

Dockerfile Reference: https://docs.docker.com/engine/reference/builder/

Build and run the image


```
$ docker build . -t bitnami/nginx:latest
Sending build context to Docker daemon  52.22kB
Step 1/17 : FROM bitnami/minideb:stretch
...
Step 17/17 : CMD [ "/run.sh" ]
 ---> Running in 6b7c35972481
Removing intermediate container 6b7c35972481
 ---> 35da09f64bf2
Successfully built 35da09f64bf2
Successfully tagged bitnami/nginx:latest


$ docker run --name nginx bitnami/nginx:latest

 11:51:30.82 Welcome to the Bitnami nginx container
...
 11:51:30.86 INFO  ==> ** Starting NGINX setup **
 11:51:30.88 INFO  ==> Validating settings in NGINX_* env vars...
 11:51:30.90 INFO  ==> Initializing NGINX...
 11:51:30.92 INFO  ==> ** NGINX setup finished! **

 11:51:30.95 INFO  ==> ** Starting NGINX **
```

We can build and run a container using **docker build** and **docker run** commands.

The images can be uploaded to a repository.

https://hub.docker.com/ is the most popular one.

Build and run the image


```
version: '2'

services:
  nginx:
    image: 'bitnami/nginx:1.16'
    ports:
      - '80:8080'
      - '443:8443'
```

We can also use **docker-compose** to run a Docker container.

Docker Compose Reference:
https://docs.docker.com/compose/

```
$ docker-compose up
Creating network "debian-9_default" with the default driver
Creating debian-9_nginx_1 ... done
Attaching to debian-9_nginx_1

nginx_1  |  11:54:16.43 Welcome to the Bitnami nginx container
...
nginx_1  |  11:54:16.46
nginx_1  |  11:54:16.47 INFO  ==> ** Starting NGINX setup **
nginx_1  |  11:54:16.48 INFO  ==> Validating settings in NGINX_* env vars...
nginx_1  |  11:54:16.49 INFO  ==> Initializing NGINX...
nginx_1  |  11:54:16.50 INFO  ==> ** NGINX setup finished! **
nginx_1  |
nginx_1  |  11:54:16.51 INFO  ==> ** Starting NGINX **
```

The containers pipeline

The containers pipeline

The container pipeline aims to automate the whole process of building, testing and publishing containers.

But.. Is that enough?

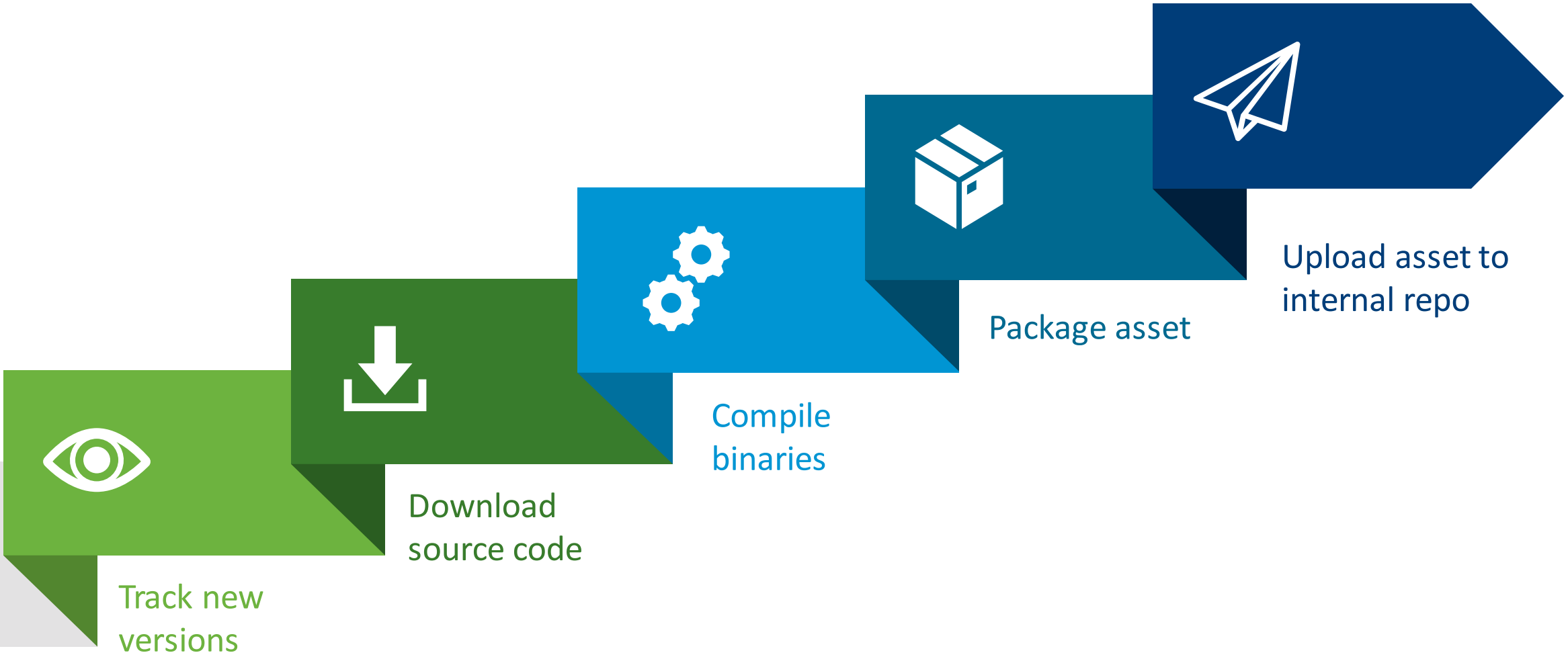
As we've seen building containers consist on writing a Dockerfile with a set of instructions and run a couple of commands.

Those instructions can run commands, set env. variables, copy files, etc… Some of those commands consist on installing **software packages** (system or "bitnami" ones).

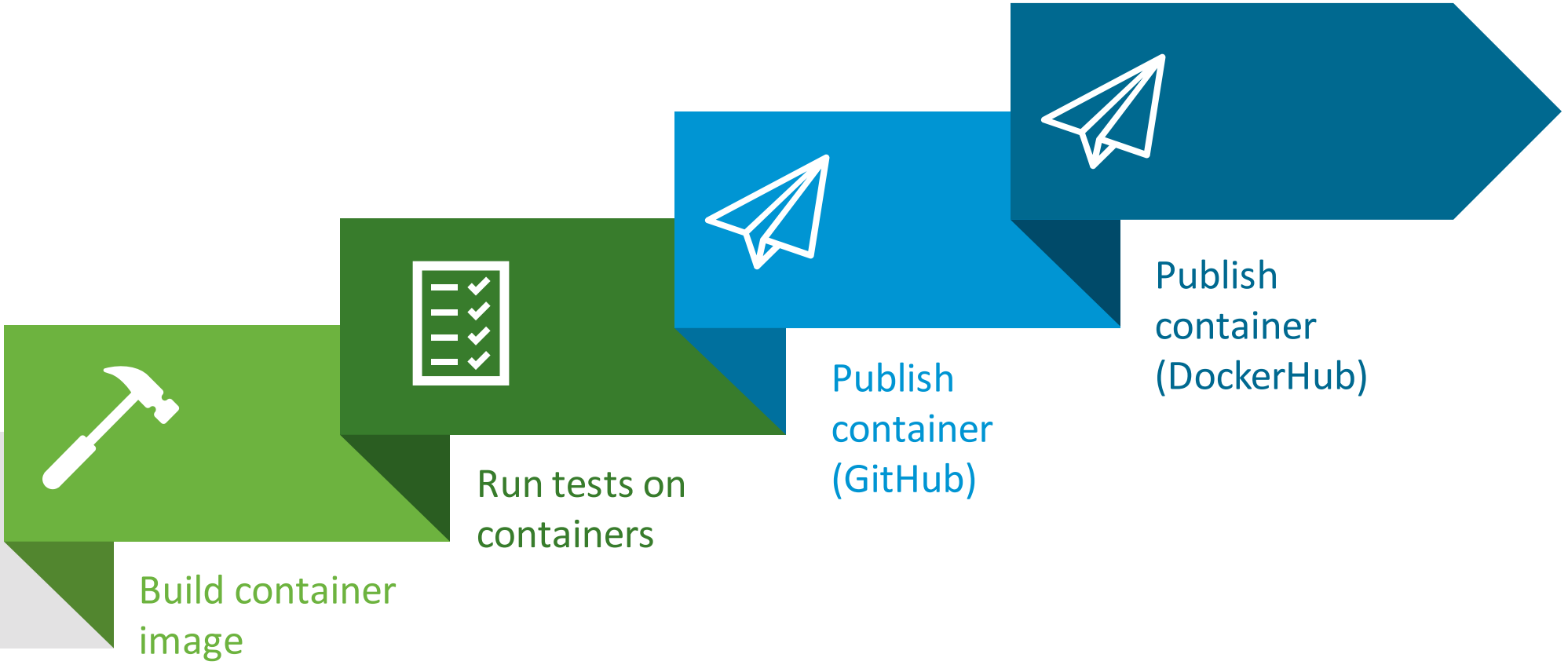
So.. Who's going to provide those packages? We **need to build them**!

... process of building, testing and publishing ~~containers~~ (containers and the packages used on them).

The containers pipeline



The containers pipeline



The containers pipeline: overview

Track new versions

- A tool called **vtracker** is used. It is run periodically (cronJob).
- The tool checks the different websites where each project announces their releases.

Download Source Code

- A tool called **downlader** is used to do so. It is run just after vtracker finishes.
- The tool downloads the source code of each project and uploads it to an internal server.

The containers pipeline: overview

Compile Binaries

- A tool called **blacksmith** retrieves the assets' source code and compile it based on some 'compilation recipes.

Package asset

- A tool called **nami** is used to create some logic that configures the asset.
- A tool called **fisherman** is used to create the package. It adds some metadata and creates a tarball with the asset's code&binaries plus the Nami logic.
- The packages are called "nami modules".

The containers pipeline: overview

Upload asset to internal repo

- The Nami modules are uploaded to an internal repo.

The containers pipeline: overview

Build container image

- A tool called **packager** is used. The packager has the ability to combine different nami modules and system packages to create Dockerfiles for each kind of solution.
- Each solution created by the packager is specified on some jsonnet manifests (a templating language).

Run tests on containers

- Containers are run and a battery of verification & functional tests are passed using a tool called **BTS** (Bitnami Test System).
- Verification tests are based on Mocha while functional ones are based on CasperJS.
- *** Also HELM ***

The containers pipeline: overview

Publish container (GitHub)

- The packager has the ability to create every file available on the GitHub repositories based on templates.
- Once the tests pass, they're pushed to GitHub.

Publish container (Dockerhub)

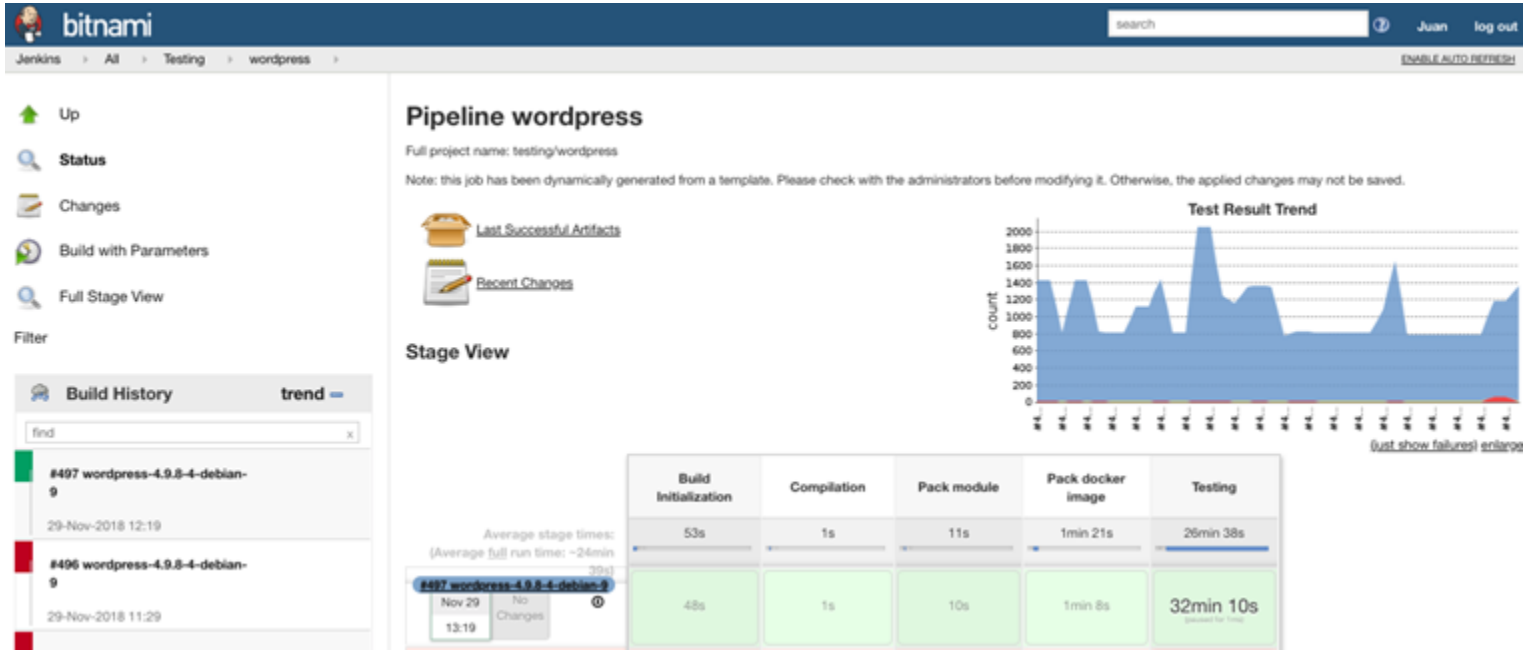
- A tool called **publisher** is used. It publishes the images on different docker repositories such as Quay.io, DockerHub or Azure Registries.

Jenkins

Okay.. I have a lot of tools that can do each of the steps necessary.. But.. is it automated??

Somebody needs to coordinate this...

The leading open source automation server.
Written in Java.
Mainly used to create CI/CD pipelines.
Website: https://jenkins.io/



Jenkins - Some numbers

- The whole pipeline is automated.
- It maintains a catalog with ~145 containers and ~160 different assets.
- No human intervention (if everything goes well).

- A “revision” is forced every day so the whole catalog is released **automatically every day!!!.**
  - Think about it twice… **Every day!!**
  - Around 145 containers are rebuilt, tested and publish every day.
    - But.. we support Debian, CentOS, and OracleLinux…
    - x3 -> 435 containers are rebuilt, tested and publish every day.
  - By building them daily we ensure there are no CVEs related to system packages.

  - Almost half of them (~60) have an associated Helm chart..
    - Wait a minute… a Chart needs a K8s cluster to be tested…
    - We create **on-demand** a K8s cluster per chart to be tested!!
      - **x3** (since we test them on AKS, EKS, GKE, IKS*, TKG* and EPKS*).
      - 180 K8s clusters are created every day just for testing!!!

Introduction to the “evolved” pipeline

We’re doing the same thing too many times…

E.g. “I want to build WordPress container”

- Build the Apache NM
- Build the PHP-FPM NM
- Build the WP NM
- Build the container -> Test it -> etc. etc.

E.g. “I want to build Drupal container”

- Build the Apache NM
- Build the PHP-FPM NM
- Build the Drupal NM
- Build the container -> Test it -> etc. etc.

Why not reuse them?

Introduction to the “evolved” pipeline

Jenkins

We’re doing the same thing too many times…

E.g. “I want to build WordPress container”

- Build the Apache NM
- Build the PHP-FPM NM
- Build the WP NM
- Build the container
- Test it -> FAILED

Then, I fix the error why it failed which was related to the Apache NM.
Let’s try again:

- Build the Apache NM
- Build the PHP-FPM NM
- Build the WP NM
- Build the container
- Test it
- Publish it

Why Should I rebuild them? They weren’t modified...

Introduction to the “evolved” pipeline

We need an “intelligent” system that only builds whatever is necessary...

During the last years we’ve been working on implementing a new pipeline which dynamically creates jobs for every specific actions whenever its’ necessary.

To do so, this new pipeline controls the global state of the system. Whenever there’s a “gap”, it will create the corresponding jobs to fill the gaps.

A new component called the **Control Plane Manager** will be in charge of checking the inventory and create the “gaps” whenever there’s something missing on it.

Demo

Any Questions

Thank You

We’re hiring!

We’ll be opening different positions during the next months (including positions for junior profiles)