FUJITSU

shaping tomorrow with you

Standardization for Container

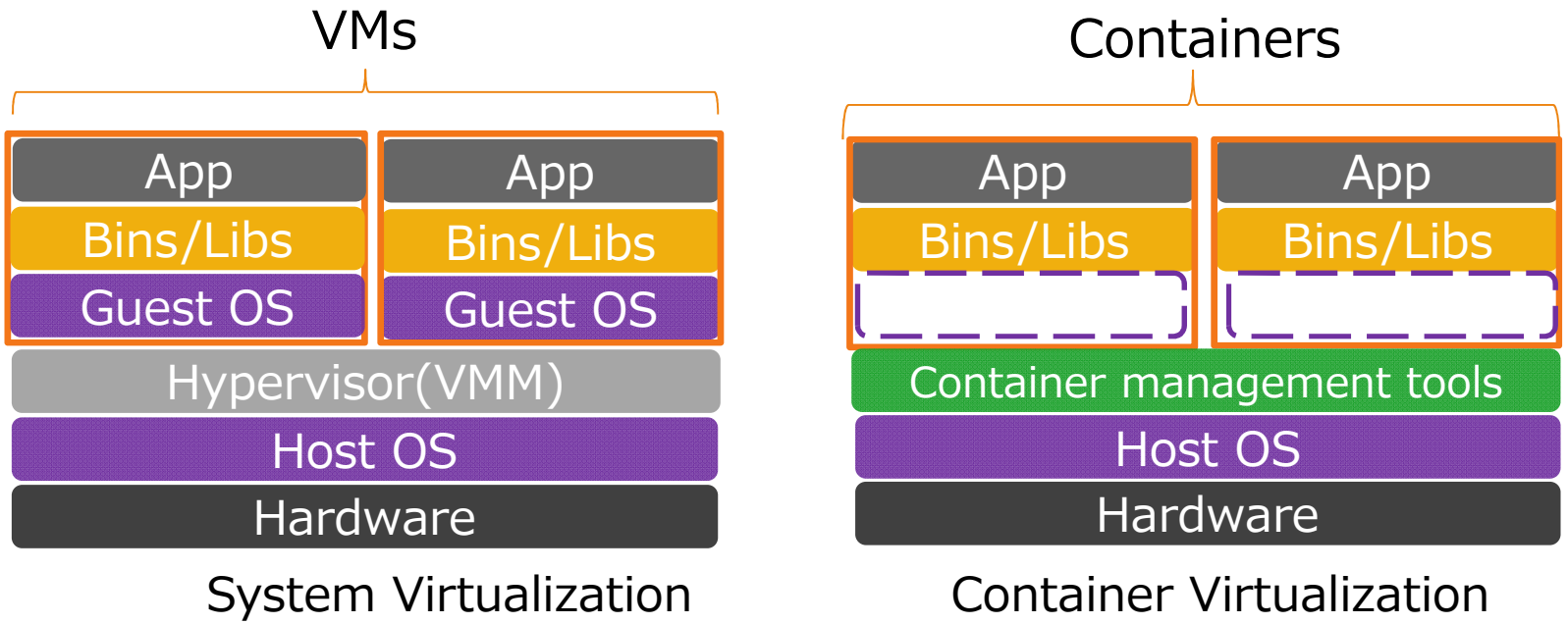
Ma Shimiao

Agenda

- Background
- Goals of container standardization
- OCI Introduction
- Current State
- Q&A

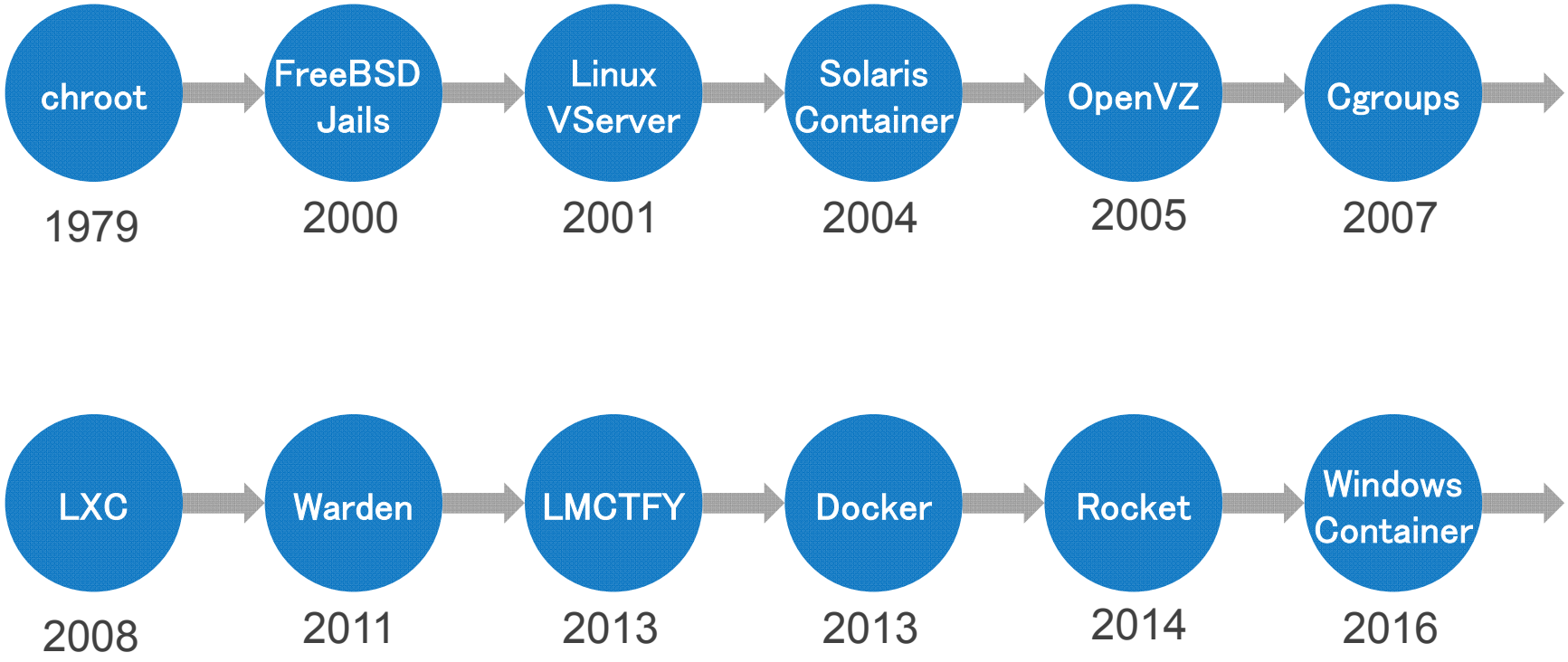
Background

- What is container
  - Operating system-level virtualization technology
  - Divide the system into and run Apps in it



Background

- Container Technology History
  - Container technology is not a new technology



Ecosystem and Containers

Background

- Many different container technologies
  - Docker
  - Rocket/rkt
  - LXD
  - Hyper
  - …
- Container-based solutions grow rapidly
  - Almost all major IT vendors and cloud providers supply
  - More and more people try to use
- There is a large ecosystem for container
  - Infrastructure vendor
  - Container runtime & orchestration

Background

But,
there is not a standard

Before A Standard

FUJITSU

- Almost everyone has its own spec
- So, container technology seems to be **fragmented**
- Users hard to choose the container providers
  - **No standards to evaluate**
  - **Not sure how to evaluate**
- Users locked into a technology vendor in the long run
  - **Hard to fit difference**
  - **High cost to transfer applications**
- Vendors also hard to choose technology
- …

Goals of Container Standardization

- Promote development of container technology
  - Unambiguous development direction
  - Platform portability issue (Unix, Linux, Solaris, Windows)
  - …
- Help container vendors to
  - Evaluate container technology
  - Choose suitable container technology
  - …
- For users
  - Guide them to choose container providers
  - Avoid being locked into any technology vendor in the long run
  - Get high quality services

Container Standardization

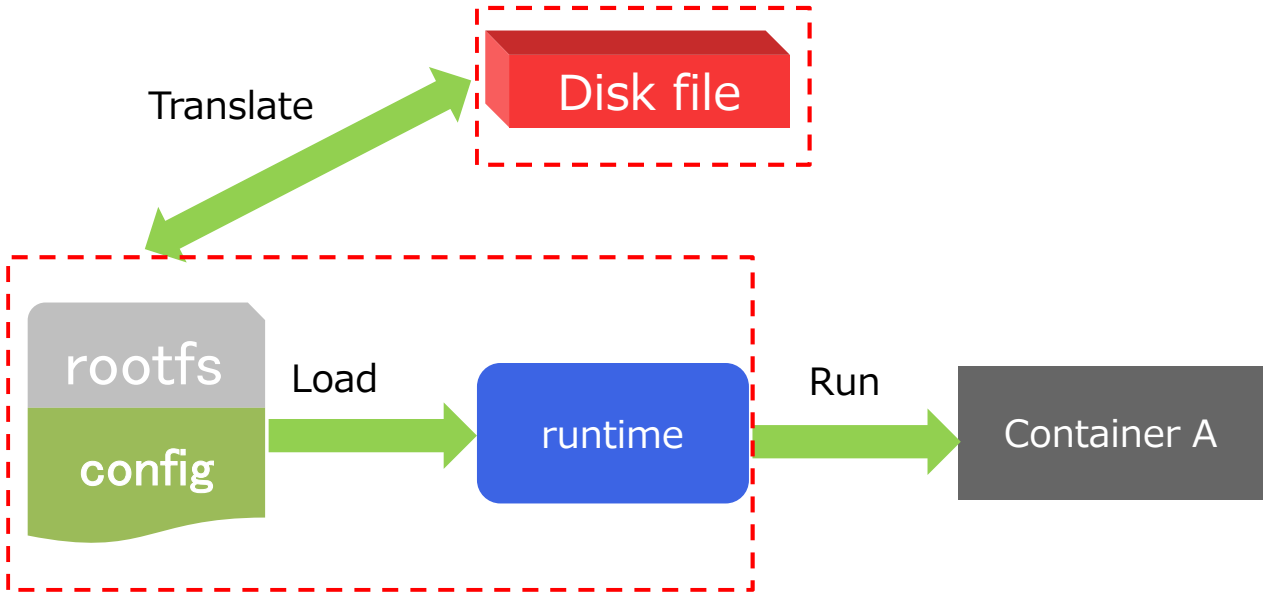
Who?

What? How?

OCI Introduction

- What is OCI
  - Open Container Initiative, launched on June 22nd 2015
  - a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation
  - 44 members, almost all major of IT vendors and cloud providers

OCI Introduction



OCI Introduction

- Container Format
  - Necessary container files for run
  - Portability
  - consume way of configuration
  - …
- Runtime Spec
  - Container lifecycle
  - Choose suitable container technology
  - …
- Container Image Format
  - Image structure
  - Necessary config items
  - …

OCI Introduction

- Mission of the OCI
  - Provides an open source community
  - Is not to define a full stack or solution requirements
  - Container format specifications and runtime
- Duties of OCI
  - Creating a formal specification for container image format and runtime
  - Accepting, maintaining and advancing the projects associated with these standards
  - Harmonizing the above-referenced standard with other proposed standards

Projects on GitHub

- runtime-spec
  - specifications for standards on Operating System process and application containers
  - http://github.com/opencontainers/runtime-spec
- runtime-tools
  - a collection of tools for working with the OCI runtime specification.
  - http://github.com/opencontainers/runtime-tools
- image-spec
  - creates and maintains the software shipping container image format spec
  - http://github.com/opencontainers/image-spec
- image-tools
  - a collection of tools for working with the OCI image specification.
  - http://github.com/opencontainers/image-tools

Runtime Spec Screenshot

FUJITSU

Filesystem Bundle

Container Format

This section defines a format for encoding a container as a *filesystem bundle* - a set of files organized in a certain way, and containing all the necessary data and metadata for any compliant runtime to perform all standard operations against it. See also MacOS application bundles for a similar use of the term *bundle*.

The definition of a bundle is only concerned with how a container, and its configuration data, are stored on a local filesystem so that it can be consumed by a compliant runtime.

A Standard Container bundle contains all the information needed to load and run a container. This MUST include the following artifacts:

1. `config.json` : contains configuration data. This REQUIRED file MUST reside in the root of the bundle directory and MUST be named `config.json` . See `config.json` for more details.
2. A directory representing the root filesystem of the container. While the name of this REQUIRED directory may be arbitrary, users should consider using a conventional name, such as `rootfs` . This directory MUST be referenced by `root` within the `config.json` file.

While these artifacts MUST all be present in a single directory on the local filesystem, that directory itself is not part of the bundle. In other words, a tar archive of a *bundle* will have these artifacts at the root of the archive, not nested within a top-level directory.

Runtime Spec Screenshot

Specification version

- `ociVersion` (string, REQUIRED) MUST be in SemVer v2.0.0 fo
  Runtime Specification with which the bundle complies. The Ope
  versioning and retains forward and backward compatibility withir
  compliant with version 1.1 of this specification, it is compatible w
  specification, but is not compatible with a runtime that supports 1

Example

```
    "ociVersion": "0.1.0"
```

Root

`root` (object, REQUIRED) specifies the container's root filesystem.

- `path` (string, REQUIRED) Specifies the path to the root filesyste
  or a relative path to the bundle. On Linux, for example, with a bu
  `/rootfs`, the `path` value can be either `/to/bundle/rootfs` or
  the field.
- `readonly` (bool, OPTIONAL) If true then the root filesystem MU
  Windows, this field must be omitted or false.

Example

```
"root": {
    "path": "rootfs",
```

Mounts

`mounts` (array of objects, OPTIONAL) specifies additional mounts
listed order. For Linux, the parameters are as documented in mount
corresponds to the 'fs' resource in the zonecfg(1M) man page.

- `destination` (string, REQUIRED) Destination of mount point:
  path.
  - Windows: one mount destination MUST NOT be nested w
  - Solaris: corresponds to "dir" of the fs resource in zonecfg(
- `type` (string, OPTIONAL) The filesystem type of the filesystem
  - Linux: valid *filesystemtype* supported by the kernel as liste
    "xfs", "reiserfs", "msdos", "proc", "nfs", "iso9660").
  - Windows: this field MUST NOT be supplied.
  - Solaris: corresponds to "type" of the fs resource in zonecfg
- `source` (string, OPTIONAL) A device name, but can also be a
  - Windows: a local directory on the filesystem of the contain
  - Solaris: corresponds to "special" of the fs resource in zone
- `options` (list of strings, OPTIONAL) Mount options of the files
  - Linux: supported options are listed in the mount(8) man pa
    specific options are listed.
  - Solaris: corresponds to "options" of the fs resource in zone

Example (Linux)

```
"mounts": [
    {
        "destination": "/tmp",
```

Runtime Spec Screenshot

FUJITSU

State

The state of a container includes the following properties:

- `ociVersion` (string, REQUIRED) is the OCI specification version use
- `id` (string, REQUIRED) is the container's ID. This MUST be unique a requirement that it be unique across hosts.
- `status` (string, REQUIRED) is the runtime state of the container. The
  - `creating` : the container is being created (step 2 in the lifecycle)
  - `created` : the runtime has finished the create operation (after step neither exited nor executed the user-specified program
  - `running` : the container process has executed the user-specified lifecycle)
  - `stopped` : the container process has exited (step 7 in the lifecycle

  Additional values MAY be defined by the runtime, however, they MUS defined above.
- `pid` (int, REQUIRED when `status` is `created` or `running`) is the
- `bundle` (string, REQUIRED) is the absolute path to the container's bu can find the container's configuration and root filesystem on the host.

Lifecycle

The lifecycle describes the timeline of events that happen fro

1. OCI compliant runtime's `create` command is invoked w identifier.
2. The container's runtime environment MUST be created unable to create the environment specified in the `confi` requested in the `config.json` MUST be created, the us time. Any updates to `config.json` after this step MUST
3. Runtime's `start` command is invoked with the unique i
4. The prestart hooks MUST be invoked by the runtime. If a the container, and continue the lifecycle at step 9.
5. The runtime MUST run the user-specified program, as s
6. The poststart hooks MUST be invoked by the runtime. If remaining hooks and lifecycle continue as if the hook ha
7. The container process exits. This MAY happen due to e being invoked.
8. Runtime's `delete` command is invoked with the unique
9. The container MUST be destroyed by undoing the steps
10. The poststop hooks MUST be invoked by the runtime. If

Projects on GitHub

- runc
  - a CLI tool for spawning and running containers according to the OCI specification
  - http://github.com/opencontainers/runc
- go-digest
  - common digest package used across container ecosystem
  - http://github.com/opencontainers/go-digest
- go-selinux
  - common SELinux package used across container ecosystem
  - http://github.com/opencontainers/go-selinux

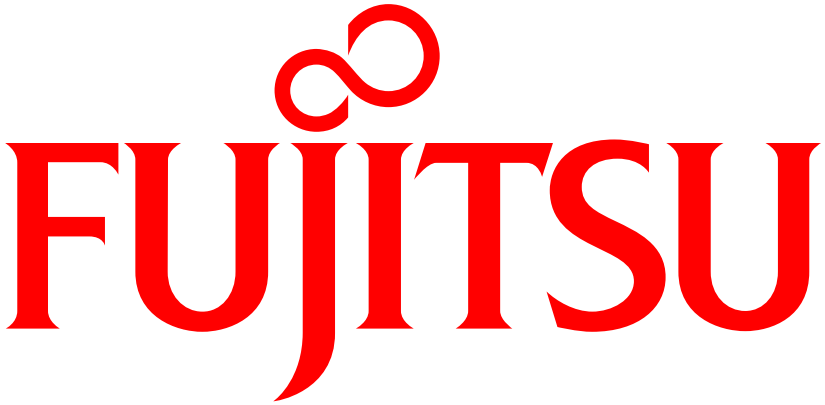
All OCI projects at https://github.com/opencontainers/

Current State

- runtime-spec v1.0 and image-spec v1.0 released
- runv, clearcontainers/runtime are compliant with spec
- certificated tools as runtime-tools and image-tools need more work

Q&A

Thank you!
Q&A



shaping tomorrow with you