

99. GSE zExpertenForum der z/OS Arbeitsgruppe

Vitznau, 23.10.2024

z/OS Container Platform

Redelf Janßen
IBM Z Brand Technical Specialist

Statement of direction

On June 23, 2020, IBM announced in

<https://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&appname=g pateam&supplier=897&letter num=ENUS220-033>

the following Statement of Direction:

- IBM intends to deliver a container runtime for IBM z/OS® in support of Open Containers Initiative compliant images comprising z/OS software.
- IBM intends to deliver Kubernetes orchestration for containers on z/OS.
- Website: <https://www.ibm.com/products/zos-container-platform>
- Product ID: 5655-MC3 (5655-MC4 S&S)

✓ z/OS Container Platform,
GA March '24

✓ z/OS Container Platform,
CD1 July '24

What is this presentation NOT about?

This presentation is **NOT** about zCX (z/OS Container Extensions).

Recall zCX is:

- Linux running in a z/OS address space
- Providing either Docker or OpenShift as the container runtime
- Allowing you to deploy Linux on IBM Z containers in that address space

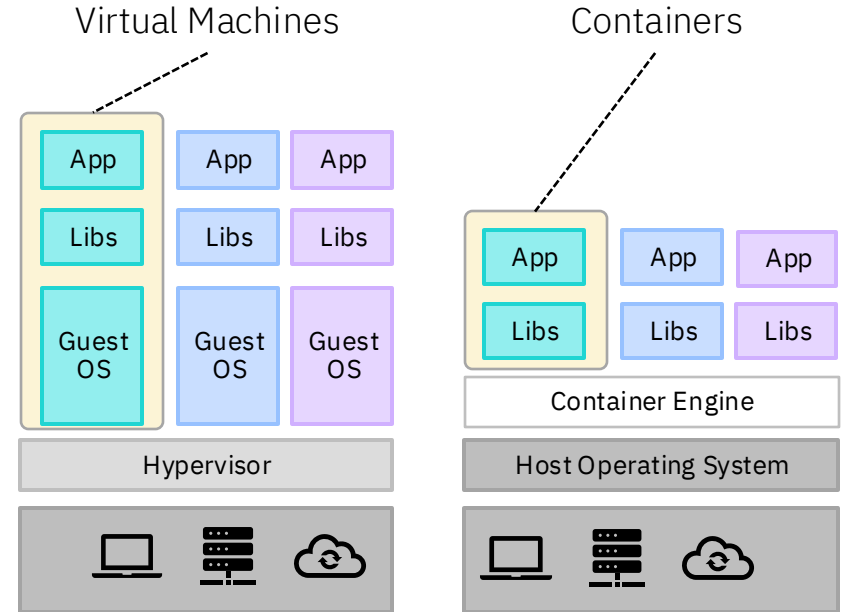
z/OS Containers is about:

- Providing a container runtime (OCI-compliant) that runs natively on z/OS
- Allowing you to build and deploy native z/OS applications as containers
- Subsequently, orchestrate those containers with a Kubernetes orchestration engine.

Comparing containers and virtual machines

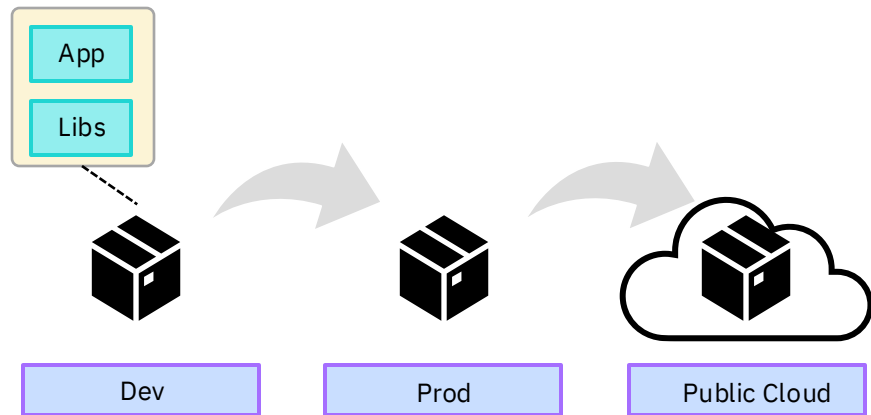
VMs are complete virtualized stacks, each running their own OS, middleware and applications. Their size is typically measured by the gigabyte.

Containers merely package the app and all the files necessary to run; they share a single instance of an operating system (OS). Container images are measured by the megabyte.

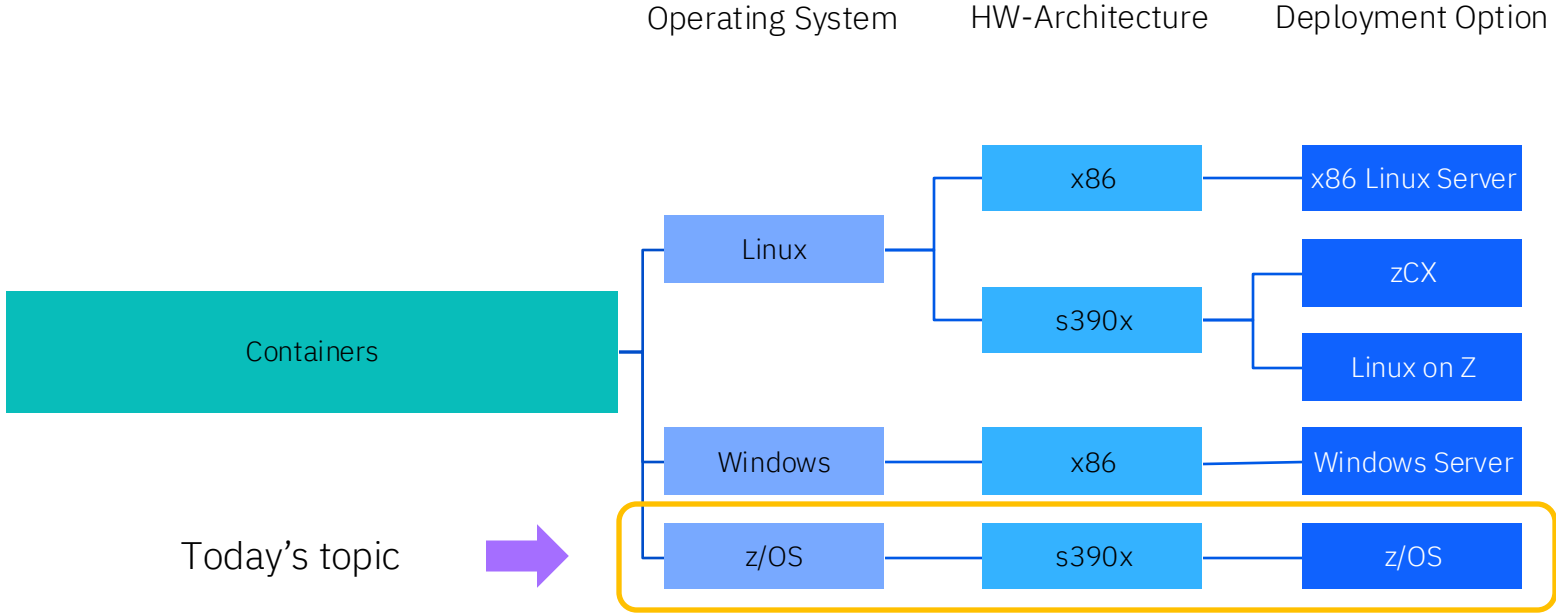


What are containers

- Containers are a [packaging mechanism](#) including all application code and required dependencies
- Containers utilize operating system facilities to run applications in isolated environments
- While containers enable decoupling of applications from the environment in which they run, they are [operating system and hardware specific](#)
- Easy and consistent deployment on different targets



Container “Genealogy”



Key benefits of containers

– skills & consistency

	<p>Standardization</p> <p>Common model for packaging and deployment of applications across the IT landscape</p>	<p>Portability</p> <p>Consistency for movement across z/OS Dev / Test / Prod</p>	
<p>Improved quality</p> <p>Containers are fully contained including application, middleware, and all dependent configuration information</p>	<p>Agility</p> <p>Deploy application and provision facilities to support agile development and meet developers' expectations</p>	<p>Isolation / Security</p> <p>Isolate applications' execution environments from each other, thereby increasing security and integrity</p>	<p>Orchestration</p> <p>Use industry standard means of deployment and management (ala Kubernetes) across a broad landscape of container-supporting platforms</p>

Containers are based on Images

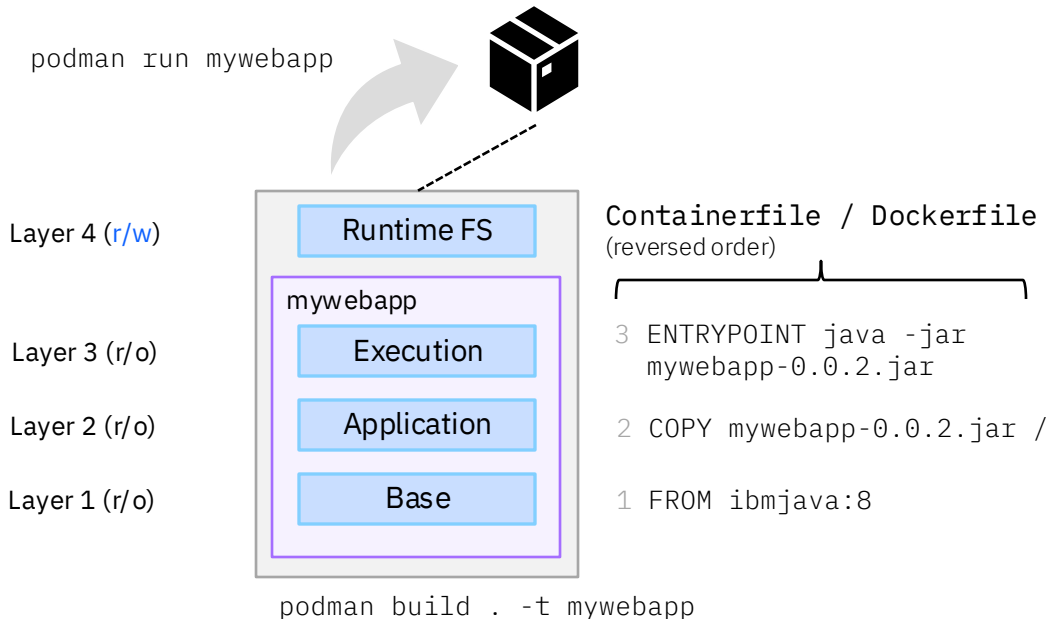
Definition

An **image** is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings

Images can only be created or read, but not modified - they are immutable

To modify an image, a new image need to be built on top of an existing image

Images become containers at runtime



What are z/OS containers?

z/OS containers [run on z/OS](#) isolated from each other through z/OS facilities

z/OS containers provide portability and consistency across z/OS LPARS and environments

Examples

- A newly developed Java / WAS (Liberty) application; will be built and deployed as a container from inception since the application developers come from a distributed background
- Aspirational: A 20-year-old mission critical COBOL application running in CICS® or IMS™ ; it can be containerized so that it can be orchestrated (via Kubernetes) with other cloud native container applications that run on non-z/OS platforms

Enables [system programmers](#) to create templates for consumption by developers in a regulated manner

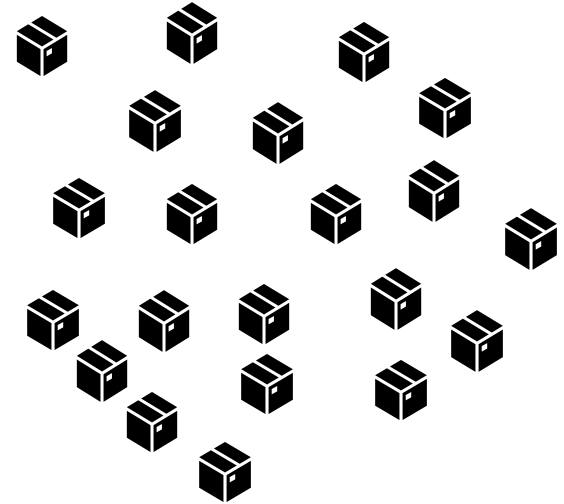
Allows [developers](#) to consume defined resources in a self-service model in an isolated, approved environment

How scalable is this?

Start small with a few containers – “ok”

Growing the size – “hmm”

Adding even more – “how do I manage this?”



A solution is needed: [Kubernetes](#) 

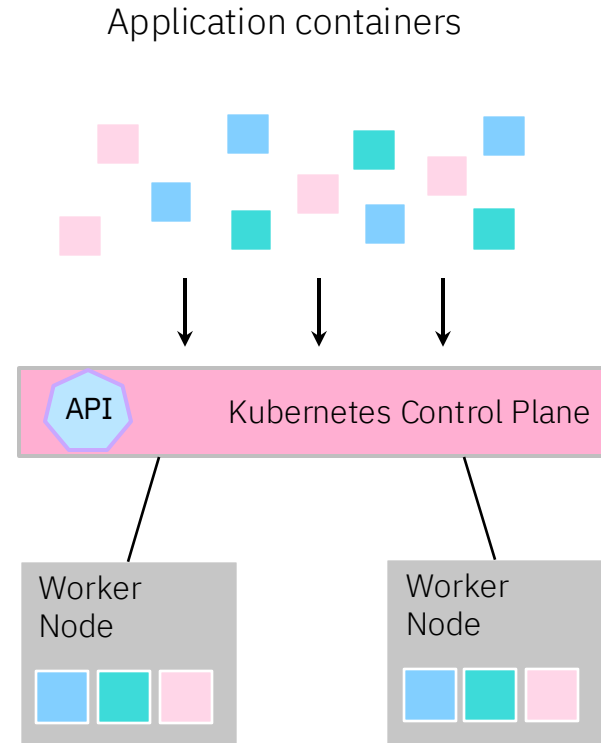
Container orchestration with Kubernetes

Kubernetes is the de facto standard container orchestration platform to manage the life cycle of containers

- Provisioning and deployment
- Availability
- Scalability
- Scheduling on infrastructure
- Health checks

Facilitates declarative management

Widely available [open-source](#) component with a large and rapidly growing ecosystem



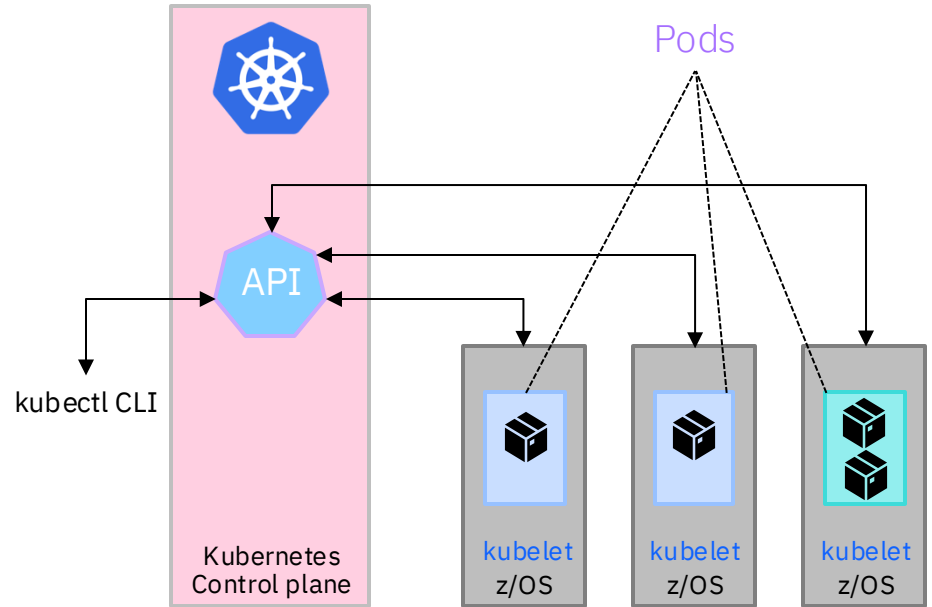
Kubernetes objects

A *Pod* is the smallest deployable unit of computing that can be created and managed in Kubernetes

It represents a single instance of an application

Typically, it consists of a single container

Sometimes it groups multiple, tightly coupled containers that share the same network and storage resources



Kubernetes objects

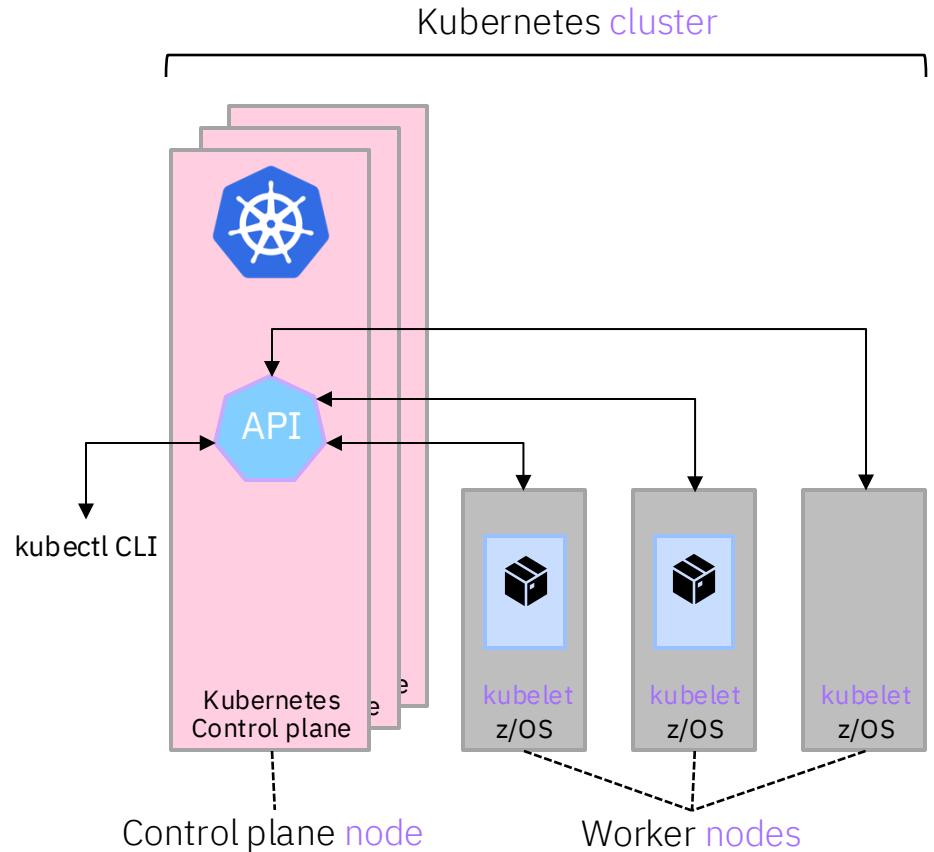
Kubernetes runs your workload by placing containers into *Pods* to run on worker machines, so called *Nodes*

A node may be a virtual or physical machine, depending on the cluster

For z/OS, a node is an LPAR or guest VM running z/OS

A *cluster* is the control plane with its components (API-server, scheduler and more) plus a collection of worker nodes

For HA-reasons, the cluster components typically run on 3 control plane nodes



Kubernetes objects

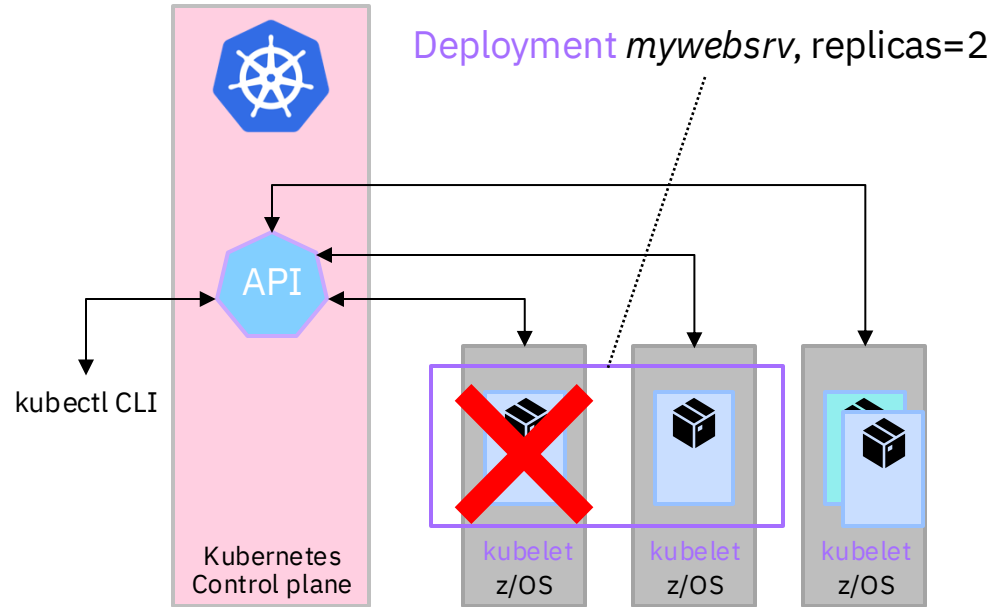
A *Deployment* manages multiple instances of an application, i.e., pods

The deployment declares how many replicas of an application should exist

You can scale the number of replicas up or down, on demand or even automatically

A deployment supports rolling updates while maintaining the availability of the application

Kubernetes manages replicas towards their desired status



z/OS approach in using open-source technologies



Formed under the Linux Foundation

OCI specifies two specifications:

- Image Specification – how to build an image
- Container Runtime – how to run an image in a container



Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications



CRI-O is an open source, community-driven container engine and is a lightweight alternative to using Docker as the runtime for Kubernetes

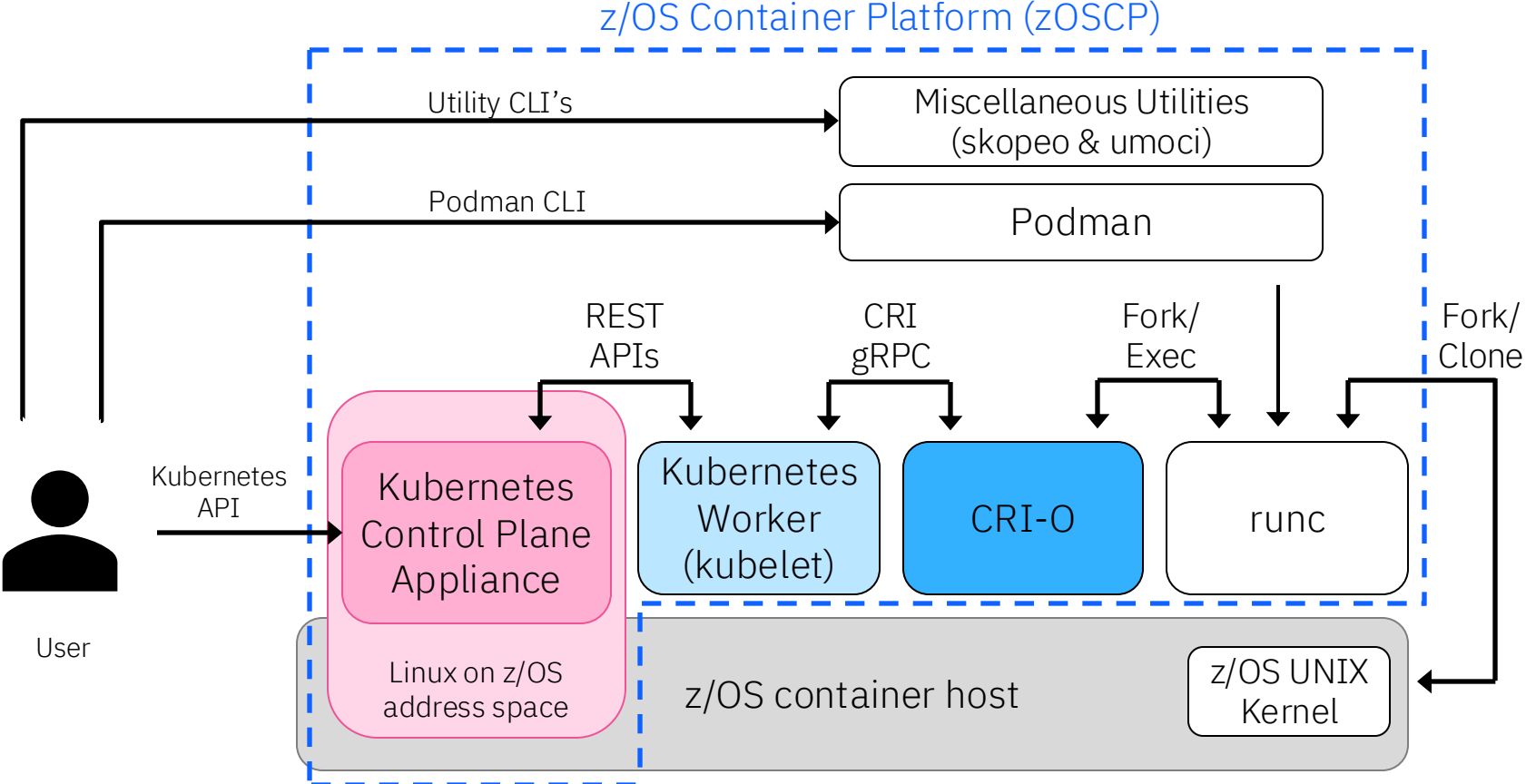


runC is a universal container runtime and a CLI for spawning and running containers according to the Open Container Initiative (OCI) specification



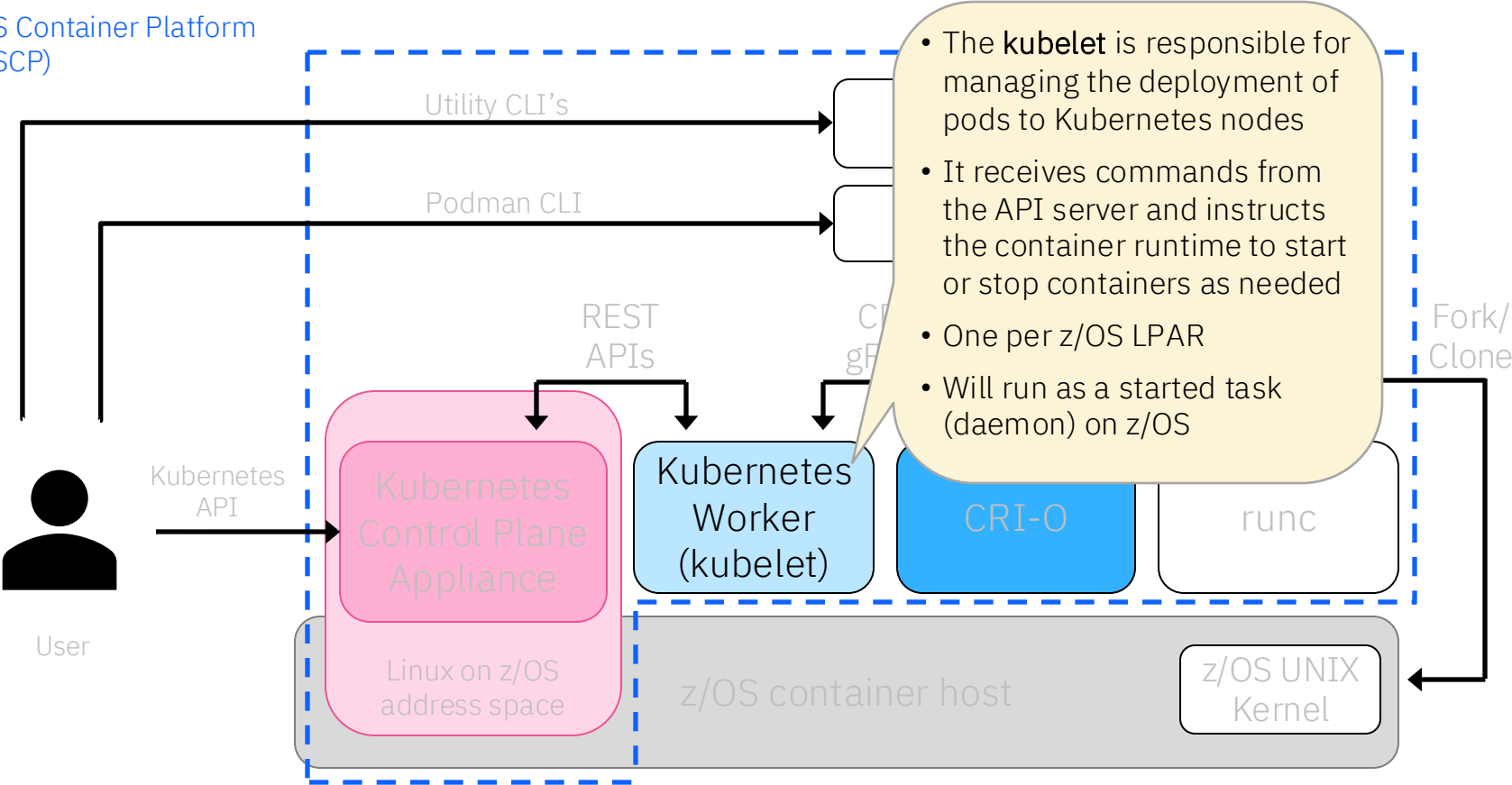
Podman (POD manager) is an open source CLI designed to make it easy to find, run, build, share and deploy applications using OCI-compliant containers and container images

z/OS Containers Architecture



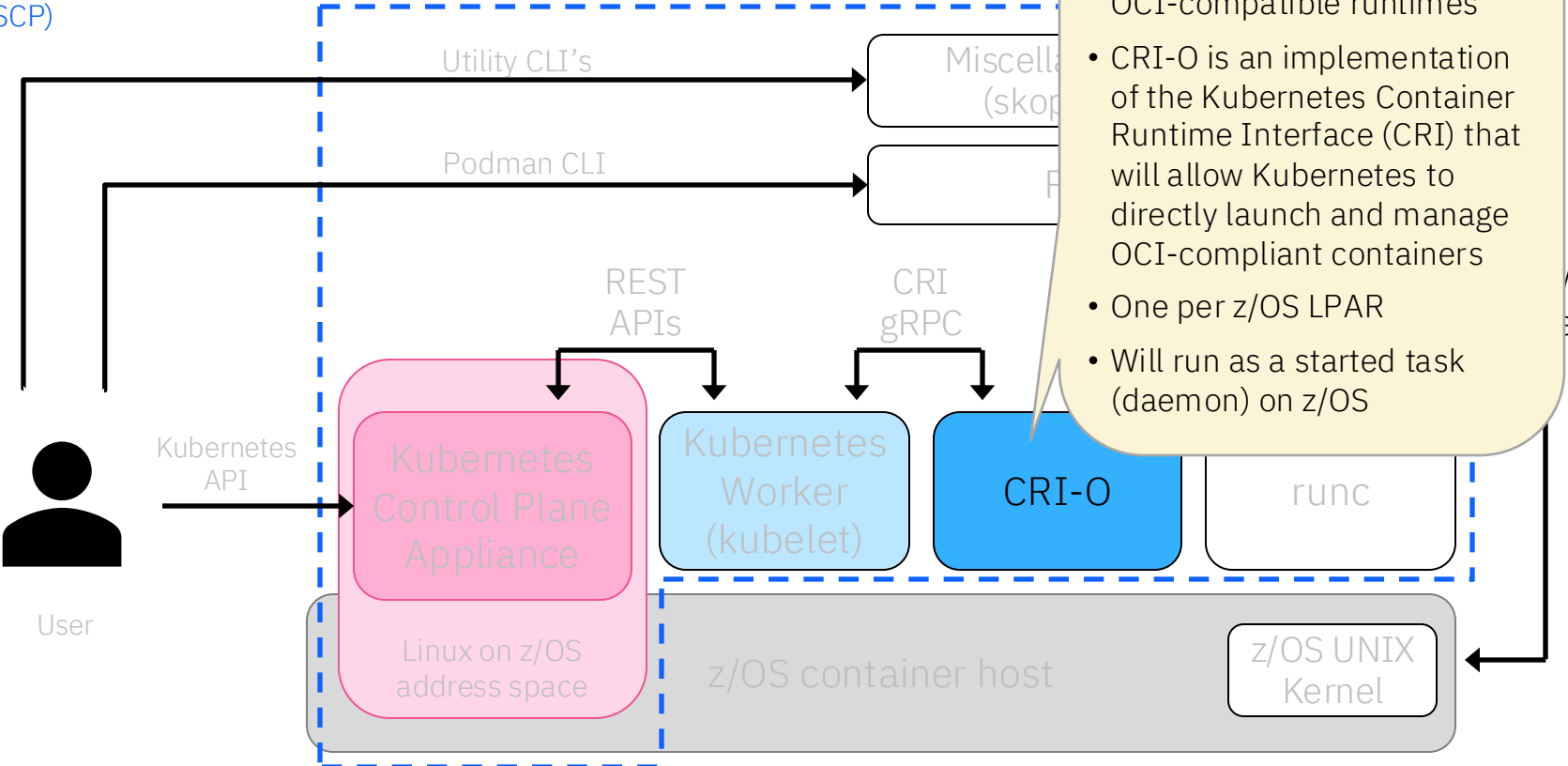
z/OS Containers Architecture

z/OS Container Platform (zOSCP)



z/OS Containers Architecture

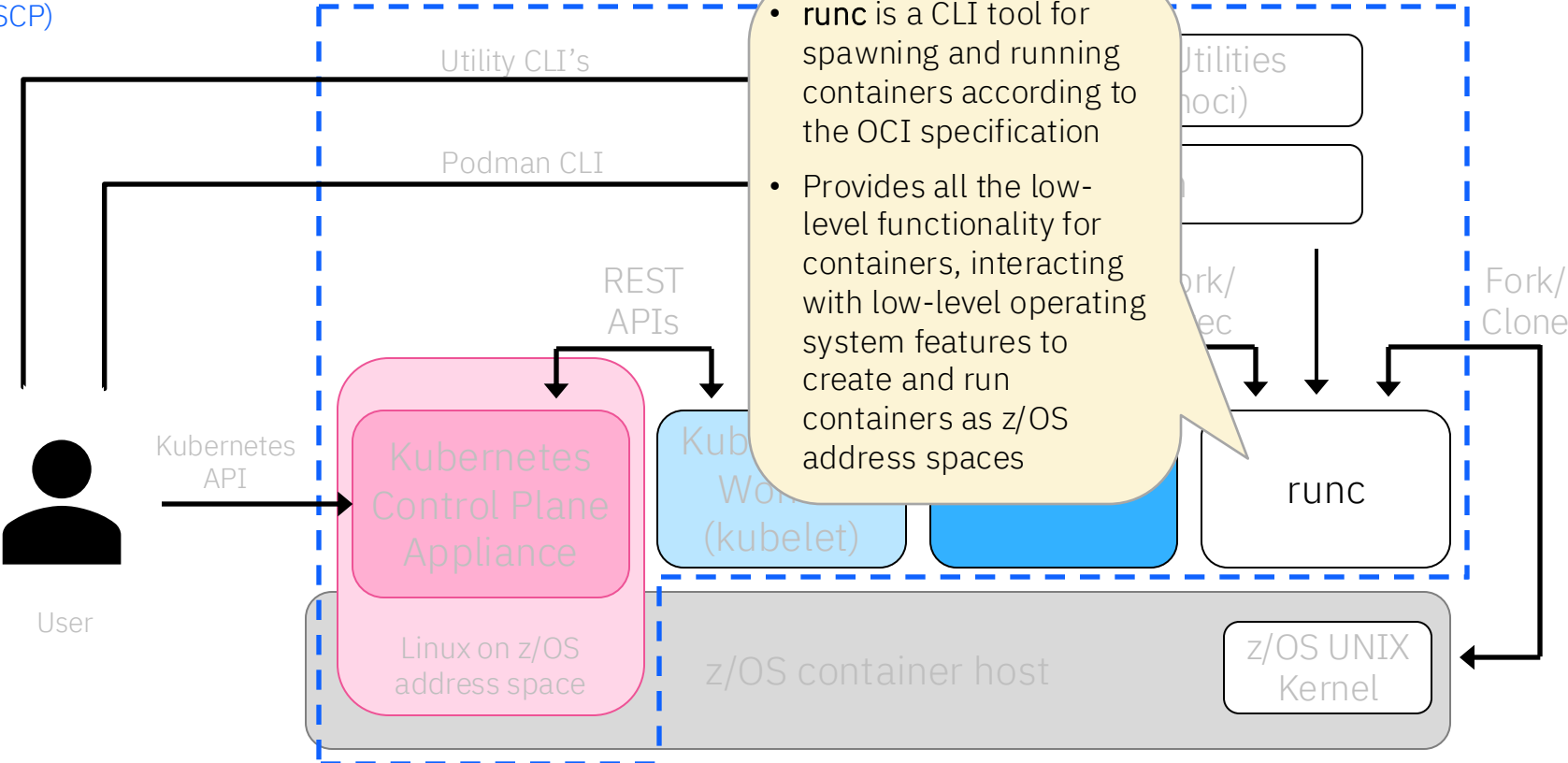
z/OS Container Platform (zOSCP)



- **CRI-O** stands for Container Runtime Interface (CRI) for OCI-compatible runtimes
- CRI-O is an implementation of the Kubernetes Container Runtime Interface (CRI) that will allow Kubernetes to directly launch and manage OCI-compliant containers
- One per z/OS LPAR
- Will run as a started task (daemon) on z/OS

z/OS Containers Architecture

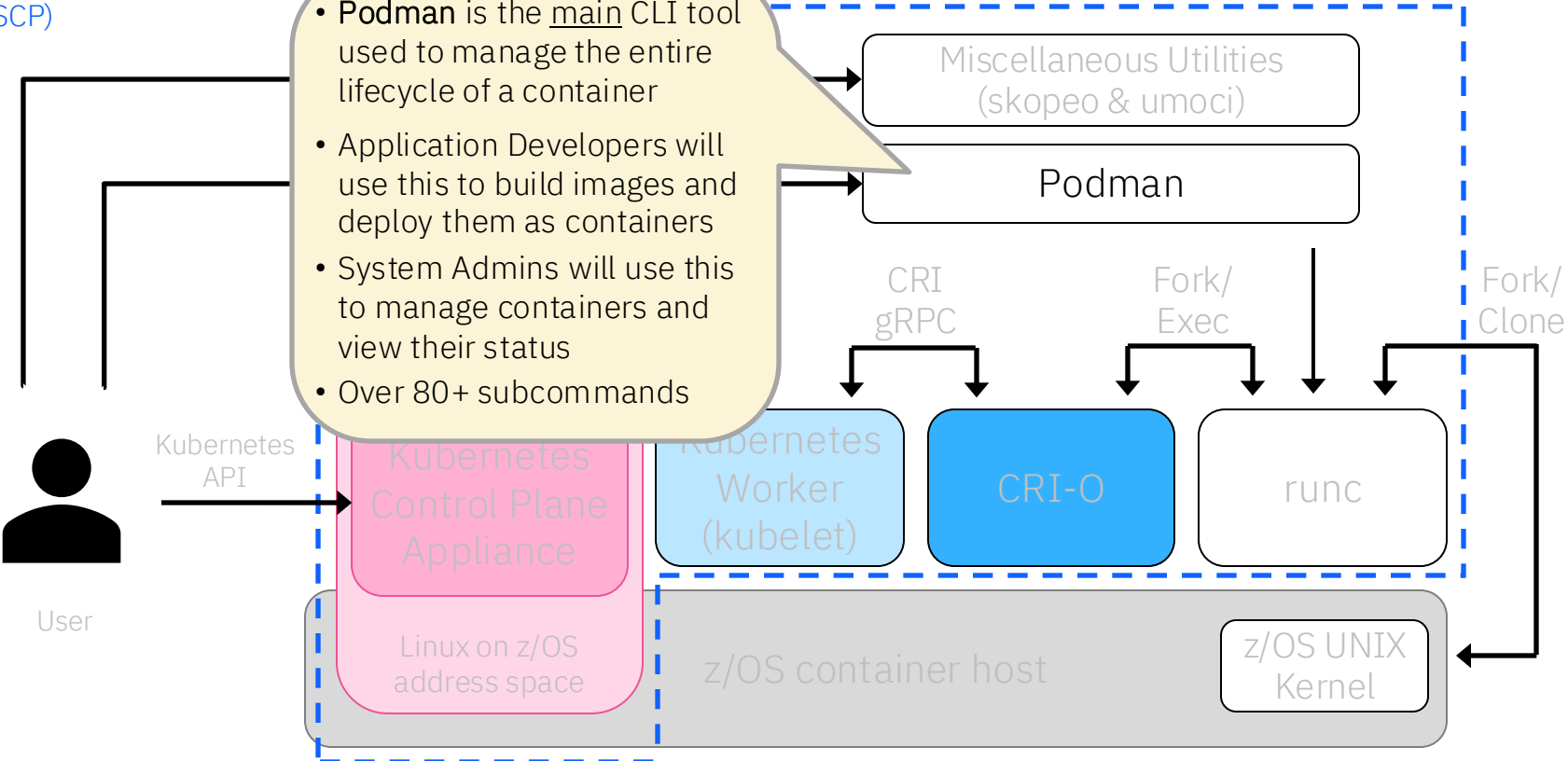
z/OS Container Platform (zOSCP)



z/OS Containers Architecture

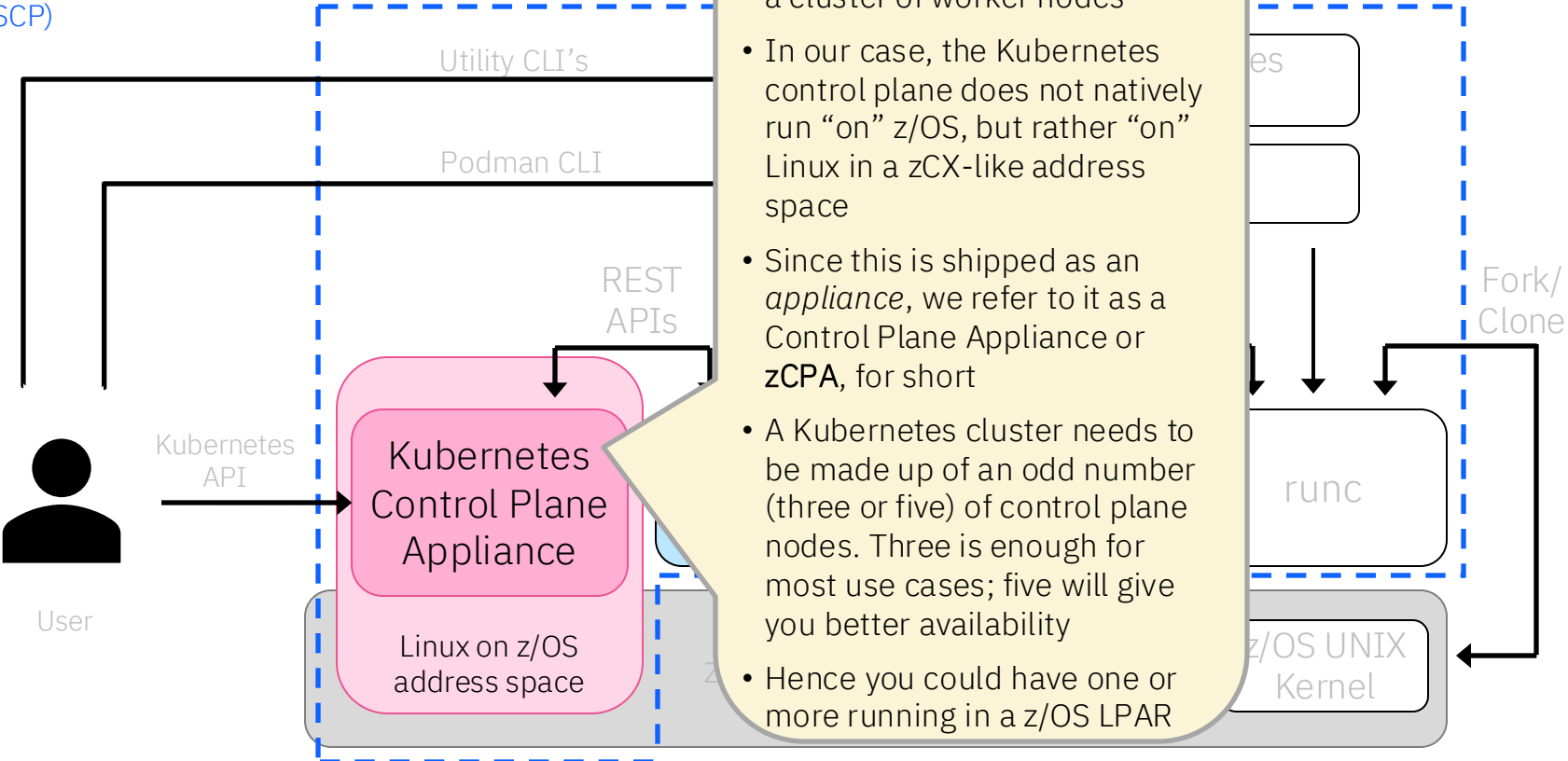
z/OS Container Platform (zOSCP)

- Podman is the main CLI tool used to manage the entire lifecycle of a container
- Application Developers will use this to build images and deploy them as containers
- System Admins will use this to manage containers and view their status
- Over 80+ subcommands



z/OS Containers Architecture

z/OS Container Platform (zOSCP)



- The **Kubernetes control plane** orchestrates containers across a cluster of worker nodes
- In our case, the Kubernetes control plane does not natively run “on” z/OS, but rather “on” Linux in a zCX-like address space
- Since this is shipped as an *appliance*, we refer to it as a **Control Plane Appliance** or **zCPA**, for short
- A Kubernetes cluster needs to be made up of an odd number (three or five) of control plane nodes. Three is enough for most use cases; five will give you better availability
- Hence you could have one or more running in a z/OS LPAR

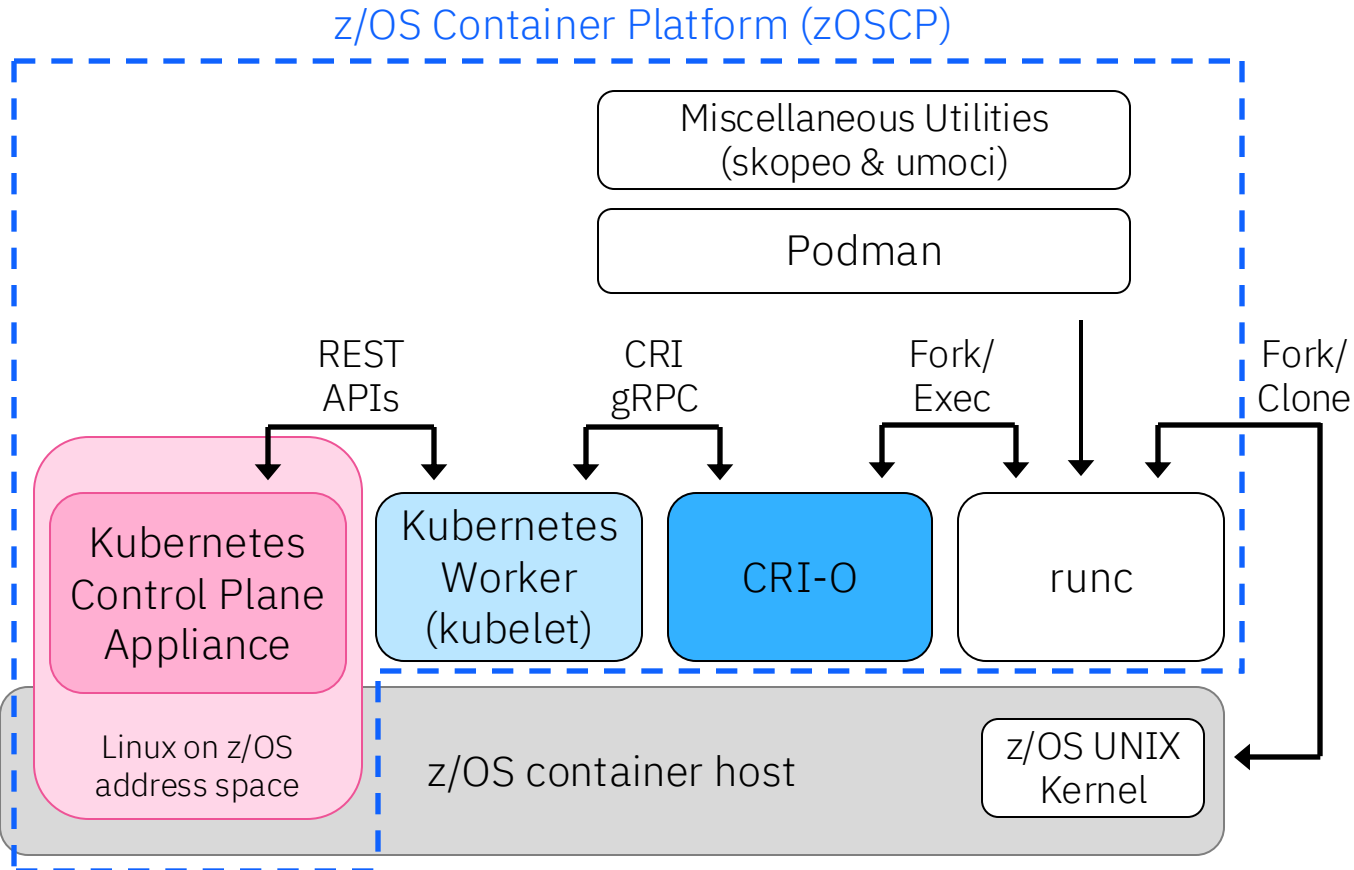
z/OS Container Platform today

July '24

z/OS 2.5 and z/OS 3.1

provide among others

- New UNIX System Services syscalls (75)
- Various enhancements to existing syscalls and C-header files
- Linux concepts of namespaces for isolation
- Kubernetes v1.29



zOSCP CD1 Support

- Available with PTFs for the following 12 APARs:
 - OA66262
 - OA66266
 - OA66267
 - OA66268
 - OA66269
 - OA66270
 - OA66361
 - OA66362
 - OA66363
 - OA66364
 - OA66365
 - OA66366

- Available since June 28, 2024

z/OS Container Dependencies provided in z/OS

New Syscalls (75)

- openat()
- openat2()
- syncfs()
- getrandom()
- getentropy()
- listxattr()
- llistxattr()
- lremovexattr()
- lsetxattr()
- removexattr()
- setxattr()
- lgetxattr()
- getxattr()
- flock()
- prlimit()
- umount2()
- nanosleep()
- Linux variant of mount()
- Linux variant of umount()
- pthread_condattr_setclock()
- Linux variant of localtime_r()
- fgetxattr()
- flistxattr()
- fremovexattr()
- fsetxattr()
- epoll_create()
- epoll_create1()
- epoll_ctl()
- epoll_pwait()
- epoll_wait()
- eventfd()
- fstatfs()
- statfs()
- dirfd()
- wait4()
- futimesat()
- utimensat()
- clock_gettime()
- futimes()
- lutimes()
- strchrnul()
- fdatasync()
- inet_aton()
- pivot_root()
- accept4()
- getline()
- sethostname()
- clone()
- unshare()
- setns()
- prctl()
- pipe2()
- dup3()
- inotify_init()
- inotify_init1()
- inotify_add_watch()
- inotify_rm_watch()
- memfd_create()
- faccessat()
- fchmodat()
- fchownat()
- fstatat()
- linkat()
- mkdirat()
- mkfifoat()
- mknodat()
- readlinkat()
- renameat()
- renameat2()
- symlinkat()
- unlinkat()
- __fchatrat()
- dprintf()
- asprintf()
- vasprintf()

Enhancements to existing syscalls & headers

- Implement BPXK_AUTOCVT for AF_LOCAL sockets
- IP_TTL flag for setsockopt() & getsockopt()
- New flags for open()
- Support “e” flag on fopen()
- New flags for socket() & socketpair()
- NAME_MAX in limit.h
- PATH_MAX in limit.h
- NI_MAXHOST in netdb.h
- NSIG in signal.h
- tm_gmtoff in time.h
- new flags on mount()
- Remount on TFS
- new flags on umount()
- Update to gethostname()
- New flags in mmap()
- PROC_SUPER_MAGIC in sys/statfs.h

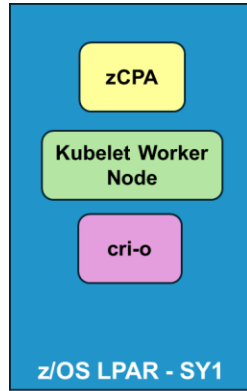
Linux Concepts

- IPC namespace
- PID namespace
- UTS namespace
- Mount namespace
- Union file system
- /proc file system
- Namespace utilities
 - nsenter
 - unshare
 - lsns

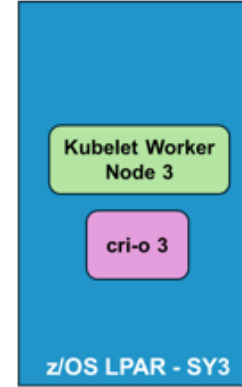
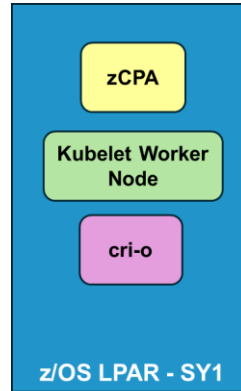
z/OS Unique Technology for Containers

- WLM support for new Service Class
- Hybrid networking support
- Update to netstat utility

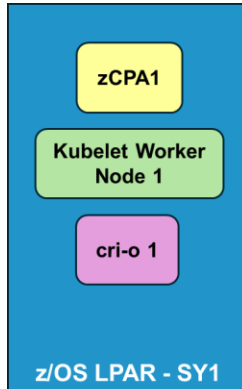
Possible IBM zOSCP Configurations 1 of 2 (including Variations thereof)



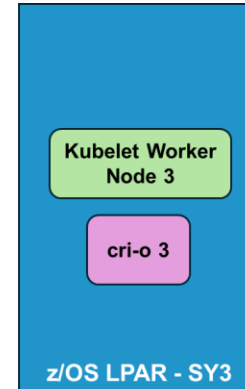
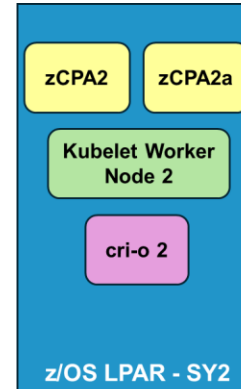
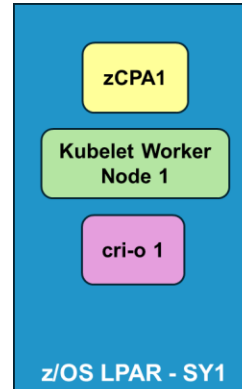
Single Instance Configuration



Single Instance Configuration+

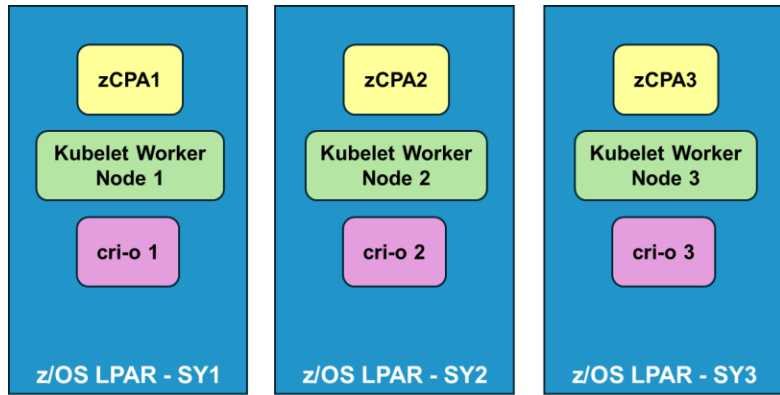


Unbalanced Configuration

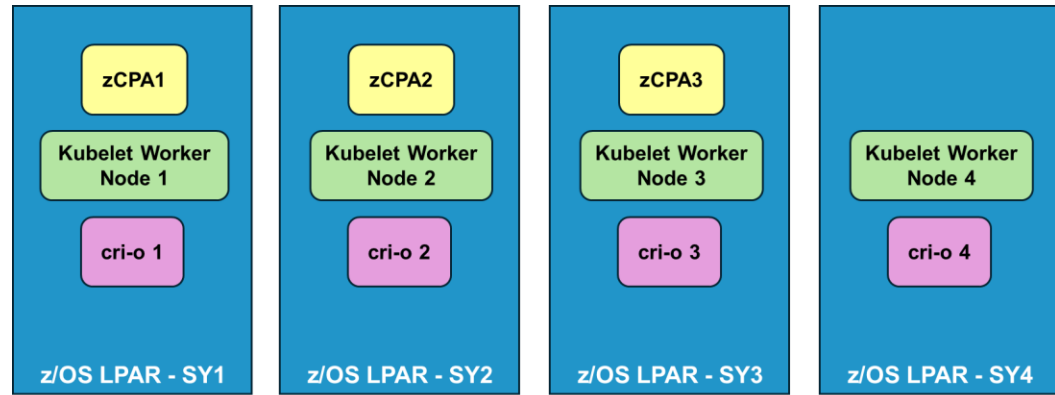


Unbalanced Configuration+

Possible IBM zOSCP Configurations 2 of 2 (including Variations thereof)



Balanced Configuration



Balanced Configuration+

Implementation

Application running in containers will be address spaces on z/OS.

z/OS has implemented the Linux concept of namespaces for isolation and virtualization. This is above and beyond the isolation already available on z/OS for address spaces.

- **PID namespace** - allows a process running inside a PID namespace to seem as if it is the only process running in that space and therefore would have a PID=1. Of course, from the outside (of the PID namespace) it will have a real PID number, but on the inside, it will have a PID of 1.
- **IPC namespace** - will allow Inter-Process Communication (IPC) to only happen between the processes running within that IPC namespace. The types of communication we are talking about are the POSIX IPC communication schemes: semaphores, shared memory and message queues. Cross-memory and common storage are not affected.

Implementation (*cont.*)

UTS namespace - UNIX Time Sharing namespace allows a process to set the hostname of that environment. This is needed because the container runtime will assign a random ID to this but can be over-ridden in the future.

Mount namespace - Essentially allows for a scoping of the file system that the processes within that mount namespace will be able to see. To establish a mount namespace, you also need to have `pivot_root()` capability (essentially another flavor of `chroot()`) whereby you are changing the root directory to a sub-portion of the file system hierarchy that has just been established. That will be file system view the process(es) will have. The other new capability that is needed for mount namespaces is bind mounts, whereby you mount things that are outside of the mount namespace file system to inside of the namespace.

Implementation (*cont.*)

Not implemented
on z/OS

- **cgroup namespaces** - We don't have cgroups on z/OS and so having a cgroup namespace on z/OS doesn't make sense. However, we have integrated with WLM, whereby one could specify a specific service they want the process (address space of a container) to run under, or if set up, we would run it in a specific WLM service class created just for containers. If nothing is set up, then we will take the lowest service class (batch).
- **Network namespace** - There is no network namespace support on z/OS. However, upon creation of a container or a pod, a dynamic VIPA is assigned to that new instance and that gives the appearance of network isolation which is very similar to what network namespaces provide on Linux.
- **User namespace** - We chose not to implement a user namespace, because we didn't want to erode the concept of “centralized security” on z/OS. We want applications running in containers to be governed by the same security database, as when applications would be running outside containers.
- We may need to invent new z/OS-specific namespaces in the future.

How to get started?

Development & Test

- Leverage an isolated self-service environment for development and test
- Employ enterprise-wide tool and process standardization enabling parallel development and continuous integration
- Spin up/down containers to introduce new features and facilitate changes

Access current base images (z/OS, Java, Golang, z/OS Connect)* from IBM Container Registry (ICR)

Create your own application image on top of a base image using [podman](#) or [umoci](#)

Use [skopeo](#) to maintain your own image registry

Containers can be started and stopped using [podman](#) from the UNIX System Services command line

* More images will be added over time

Installation and Configuration

Overview

- Workflow is provided to perform configuration of zOSCP:
 - /usr/lpp/IBM/zoscp/workflows/zoscp_general.xml
- Documentation references workflow – it is expected that the end user go through the workflow for configuration.
- Workflow is scoped at a system level and should be performed on each system where zOSCP is to be configured.
- Some steps are manual and require updates to be made to parmlib or submit a security sample job.
- Expectation is the workflow is run by a uid=0 user or one that has access to BPX.SUPERUSER.

z/OSMF Workflow

- Steps are expected to be completed in order, with some steps disabled until prior steps are completed.
- Some steps use inline JCL, while others call a shell script with BPXBATCH

Workflows > Performs the base setup for running zOSCP - Workflow_0

Performs the base setup for running zOSCP - Workflow_0

Workflow Details

Workflow Steps

Actions ▾

↔ No filter applied

<input type="checkbox"/>	State Filter	No. Filter	Title Filter
<input type="checkbox"/>	➔ Ready	1	■ Validate your BPXPRMxx pamlib is setup for zOSCP
<input type="checkbox"/>	➔ Ready	2	■ Setup SMFLIMxx for zOSCP
<input type="checkbox"/>	➔ Ready	3	■ Add the zOSCP product directory to PATH= environment variable in /etc/profile
<input type="checkbox"/>	➔ Ready	4	■ Perform podman security setup
<input type="checkbox"/>	➔ Ready	5	■ Setup permanent filesystem used by podman
<input type="checkbox"/>	✗ Not Ready	6	■ Setup sharing of Podman Filesystem
<input type="checkbox"/>	➔ Ready	7	■ Setup temporary filesystems used by containers
<input type="checkbox"/>	➔ Ready	8	■ Copy default configuration files provided for containers
<input type="checkbox"/>	➔ Ready	9	■ Establish affinity to a TCP/IP stack for z/OS UNIX Common INET (CINET) environments
<input type="checkbox"/>	➔ Ready	10	■ Perform appropriate setup for container networking
<input type="checkbox"/>	➔ Ready	11	■ Setup the WLM Service Class Policy
<input type="checkbox"/>	✗ Not Ready	12	■ Validate container setup

Parmlib Changes (steps 1-2)

- zOSCP requires UFS, TFS, and PROC filesystems set in BPXPRMxx
- /proc needs to exist before PROC can be mounted
- SMFLIMxx needs to be updated to support caches mapped about the 2GB address range

Workflows > Performs the base setup for running zOSCP - Workflow_0 > 1. Validate your BPXPRMxx parmlib is setup for zOSCP

Properties for Workflow Step 1. Validate your BPXPRMxx parmlib is setup for zOSCP

General	Details	Dependencies	Notes	Perform	Status	Input Variables	Feedback
<p>Review Instructions</p> <p>Review and confirm the instructions provided below have been performed on LOCAL.SY1, then clic</p> <hr/> <p>Instructions:</p> <p>zOSCP requires the UFS, TFS and PROC filesystems.</p> <p>To activate the union filesystem, add the following to your BPXPRMxx PARMLIB member:</p> <pre>FILESYSTYPE TYPE(UFS) ENTRYPOINT(BPXUNINT)</pre> <p>To activate the temporary filesystem, add the following to your BPXPRMxx PARMLIB member:</p> <pre>FILESYSTYPE TYPE(TFS) ENTRYPOINT(BPXTFS)</pre> <p>To activate the PROC filesystem, add the following to your BPXPRMxx PARMLIB member:</p> <pre>FILESYSTYPE TYPE(PROC) ENTRYPOINT(BPXUPINT)</pre>							

Workflows > Performs the base setup for running zOSCP - Workflow_0 > 2. Setup SMFLIMxx for zOSCP

Properties for Workflow Step 2. Setup SMFLIMxx for zOSCP

General	Details	Dependencies	Notes	Perform	Status	Input Variables	Feedback
<p>Review Instructions</p> <p>Review and confirm the instructions provided below have been performed on LC</p> <hr/> <p>Instructions:</p> <p>There is a storage requirement to support caches mapped above the 2 GB ad above the 2 GB address range, by specifying the following in your SMFLIMxx</p> <pre>REGION JOBNAME(*) MAXSHARE(262144)</pre> <p>To modify SMFLIMxx PARMLIB settings without reloading the initial program, :</p>							

PATH update (step 3)

- The `/usr/lpp/IBM/zoscp/bin` should be added to the PATH environment variable in `/etc/profile`

The screenshot displays a web-based workflow interface. At the top, a blue header bar contains the word "Workflows". Below this, a breadcrumb trail reads "Workflows > Performs the base setup for running zOSCP - Workflow_0 > 3. Add the zOSCP product directory to PATH= environment variable in /etc/profile". The main heading is "Properties for Workflow Step 3. Add the zOSCP product directory to PATH= environment variable in /etc/profile". A navigation bar includes tabs for "General", "Details", "Dependencies", "Notes", "Perform" (which is active), "Status", "Input Variables", and "Feedback". The "Perform" tab shows a "Review Instructions" section with a yellow arrow icon. The instructions text reads: "Review and confirm the instructions provided below have been performed on LOCAL.SY1, then click **Finish** to mark the step complete." Below this, a section titled "Instructions:" contains the following text: "Add `/usr/lpp/IBM/zoscp/bin` to your PATH environment variable in `/etc/profile` as like the example below:" followed by a code block:

```
# Add zoscp executables to PATH
PATH=$PATH:/usr/lpp/IBM/zoscp/bin
```

Security Setup (step 4)

- This step points to the sample job SYS1.SBCZSMPL(BCZSECS1) which needs to be reviewed, modified and run by a z/OS Security Administrator.
- The security job creates a PODMAN group and gives it access to the CONTAINERS SAF resource in the UNIXPRIV class.
- This gives non-UID 0 users access to run containers on z/OS with tools like podman.

```
/* Create group that will be given access to run podman          */
/* ADDGROUP PODMAN OMVS(AUTOGID)                                */
/* RDEFINE UNIXPRIV CONTAINERS UACC(NONE)                        */
/* PERMIT CONTAINERS ID(PODMAN) ACCESS(READ) CLASS(UNIXPRIV)   */
/* SETROPTS RACLIST(UNIXPRIV) REFRESH                           */

/* To connect a user ID to the PODMAN group, run:              */
/* CONNECT <userid> GROUP(PODMAN)                               */
/* Where the <userid> is the user ID you want to connect to the */
/* PODMAN group.                                               */
```

Filesystem Setup (steps 5,6,7)

- These automated steps setup both permanent and temporary filesystems used by podman:
 - `/var/lib/podman` : permanent filesystem used by an administrator to store images for other users of podman
 - `/var/run/containers` : tfs used for container metadata for uid=0
 - `/var/run/runc` : tfs used by runc for uid=0
 - `/var/run/user` : tfs where user specific metadata is stored
- After `/var/lib/podman` is setup, a bind mount is used to share `/var/lib/podman/storage` read-only with unprivileged users. The bind mount is created at `/var/share/containers/storage`.
- `/var/share/containers/storage` is the default additional image store for unprivileged users on z/OS.

Configuration Files (step 8)

- This step copies default container configuration files to the appropriate location on the system:
 - `containers.conf` : default configuration options for podman
 - `mounts.conf` : default mounts for podman
 - `registries.conf` : container registry configuration
 - `storage.conf` : container storage options
 - `policy.json` : specifies policy for accepting images. Note that the default we provide rejects images from all registries. The podman trust command can be used to trust a registry, which will update this file and allow images to be pulled from that location.
- If these files already have been copied, no copy is done.

Networking Setup (step 9,10)

- These steps go over the configuration updates required to support container networking on zOSCP.
- VIPARANGE ZCONTAINER IP address ranges need to be setup manually in the TCP/IP profile statement.

The screenshot displays the 'Workflows' interface for step 10, 'Perform appropriate setup for container networking'. The interface includes a navigation bar with 'Workflows' and 'Settings | Help'. Below the navigation bar, the title is 'Properties for Workflow Step 10. Perform appropriate setup for container networking'. The main content area has tabs for 'General', 'Details', 'Dependencies', 'Notes', 'Perform', 'Status', 'Input Variables', and 'Feedback'. The 'Perform' tab is active, showing a list of steps: 'Input Variables' (checked), 'General' (checked), 'Review Instructions' (highlighted with a yellow arrow), 'Create JOB statement', 'Review JCL', and 'Submit and Save JCL'. The 'Review Instructions' section contains the following text:

Review Instructions

To proceed with the guided path through creating and submitting the JCL on LOCAL.SY1, click **Next**. Or, you can choose to bypass this step. If so, first review and confirm that the instructions below are complete. Then, click **Finish** to mark the step complete.

Instructions:

CNI plugins are shipped in the directory `/usr/lpp/IBM/zoscp/bin` which are used to support networking for zOSCP. This workflow step will create a copy of the IBM provided configuration file that uses the CNI plugin. By default, IPv4 networking is enabled in the CNI configuration. If IPv6 networking is desired, then select IPv6 in the drop down to enable IPv6 networking in the CNI configuration. Note that this will disable IPv4 networking. You can update this later on by changing the IPv4 and IPv6 settings in the configuration file to true or false.

The configuration file that will be created is `/etc/cni/net.d/10-zoscni.conflist`

In addition to performing this workflow step, you will also need to configure a range of IP addresses for zOSCP. Refer to the **VIPARANGE ZCONTAINER TCP/IP** profile statement in the z/OS Communications Server: IP Configuration Reference.

This step should be run from a user that has a UID of 0 or has access to the SAF resource BPX.SUPERUSER in the FACILITY class.

WLM Configuration (step 11)

- This is a manual step that provides instructions on how WLM can be used to classify zOSCP workloads.
- The SYSCNTNR service class or a container qualifier can be used to classify the work.

The screenshot shows the IBM Workflows interface for step 11: Setup the WLM Service Class Policy. The breadcrumb trail is: Workflows > Performs the base setup for running zOSCP - Workflow_0 > 11. Setup the WLM Service Class Policy. The page title is "Properties for Workflow Step 11. Setup the WLM Service Class Policy". The "Perform" tab is selected, and the "Review Instructions" section is active. The instructions state: "WLM can be used to classify zOSCP workloads. By default, zOSCP work will run in service class SYSOTHER, which has a discretionary goal. If this is not appropriate for your environment, you can use WLM to create the SYSCNTNR service class or use a container qualifier to classify the work. Creating the SYSCNTNR service class can be useful to ensure that zOSCP work, by default, does not get assigned to the SYSOTHER service class. Using a container qualifier can be useful if you have multiple sets of zOSCP work with different requirements - a descriptive name can be given to each group of requirements."

Workflows

Workflows > Performs the base setup for running zOSCP - Workflow_0 > 11. Setup the WLM Service Class Policy [Setting](#)

Properties for Workflow Step 11. Setup the WLM Service Class Policy

General Details Dependencies Notes **Perform** Status Input Variables Feedback

Review Instructions

Review Instructions

Review and confirm the instructions provided below have been performed on LOCAL.SY1, then click Finish to mark the step complete.

Instructions:

WLM can be used to classify zOSCP workloads.

By default, zOSCP work will run in service class SYSOTHER, which has a discretionary goal. If this is not appropriate for your environment, you can use WLM to create the SYSCNTNR service class or use a container qualifier to classify the work.

Creating the SYSCNTNR service class can be useful to ensure that zOSCP work, by default, does not get assigned to the SYSOTHER service class. Using a container qualifier can be useful if you have multiple sets of zOSCP work with different requirements - a descriptive name can be given to each group of requirements.

Installation Verification (step 12)

- The final step of the workflow runs an installation verification program (shell script)
- The program does the following checks:
 - Checks to make sure all necessary filesystems are mounted and enabled (UFS, TFS, PROC)
 - Checks to make sure install directories exist (/usr/lpp/IBM/zoscp/bin)
 - Ensures LE has been setup appropriately to run containers
 - Builds and runs a podman “hello world” rexx image
- This program can also be run in the shell outside of the workflow.

Images and Image Management

Overview

- Problem Statement / Need Addressed / User Stories:
 - Customers need a “base” environment in their containers
 - z/OS Container Platform target image infrastructure requires special authority
- Solution:
 - IBM provides a “base” image to customers through IBM Cloud Container Registry
- Benefit / Value:
 - Use of “base” image and icr.io is existing/familiar and aids build/debug
 - z/OS administrator continues to control security characteristics of system

Image architecture and operating system

- Every OCI image has an architecture and os

```
$ skopeo inspect --config docker://icr.io/zos
{
  "created": "2024-02-15T22:57:27.624488767Z",
  "architecture": "s390x",
  "os": "zos",
```

- z/OS Container Platform uses **zos** on **s390x** images, *not linux images*
- Prior to z/OS Container Platform, such images did not exist

Image Locations

Image name	Location	Description
<i>zos</i>	<code>icr.io/zoscp/zos:latest</code>	Foundation image, consisting of a basic z/OS UNIX environment with core z/OS programs and libraries.
<i>ibmjava</i>	<code>icr.io/zoscp/ibmjava:8</code>	Builds on the z/OS base image to provide IBM SDK for z/OS, Java Technology Edition, Version 8 - 64-bit version and source code for the sample Java application.
<i>ibm-zcon-server</i>	<code>icr.io/zosconnectunlimited/ibm-zcon-server:3.0.78</code>	Builds on z/OS base and Java images to enable building a z/OS Connect application.
<i>golang</i>	<code>icr.io/zoscp/golang:latest</code>	Go (golang) is a general purpose, higher-level, imperative programming language.

location, key, and sample command provided in associated product memo

zOSCP z/OS Control Plane Appliance (zCPA)

Overview

- Problem Statement:

Need to provide control plane node for z/OS-based Kubernetes cluster

- Solution:

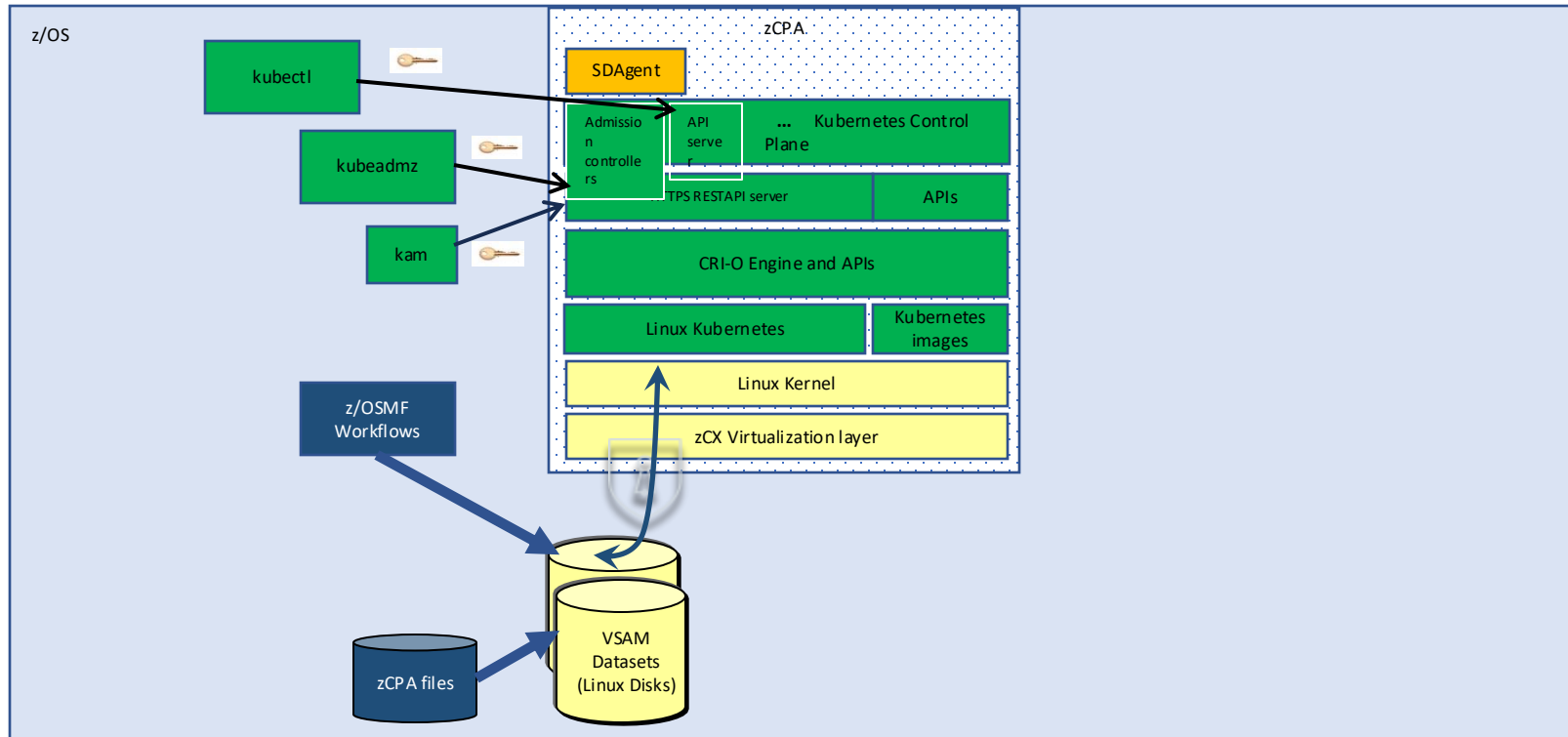
Create Linux-based appliance that runs within z/OS address space that can host the control plane node infrastructure

- Benefit / Value:

Ensure all Kubernetes cluster nodes for zOSCP reside on z/OS systems

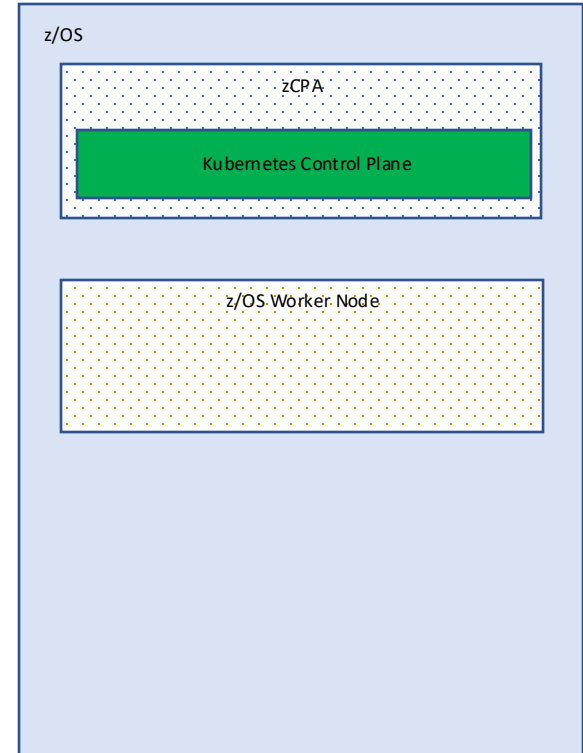
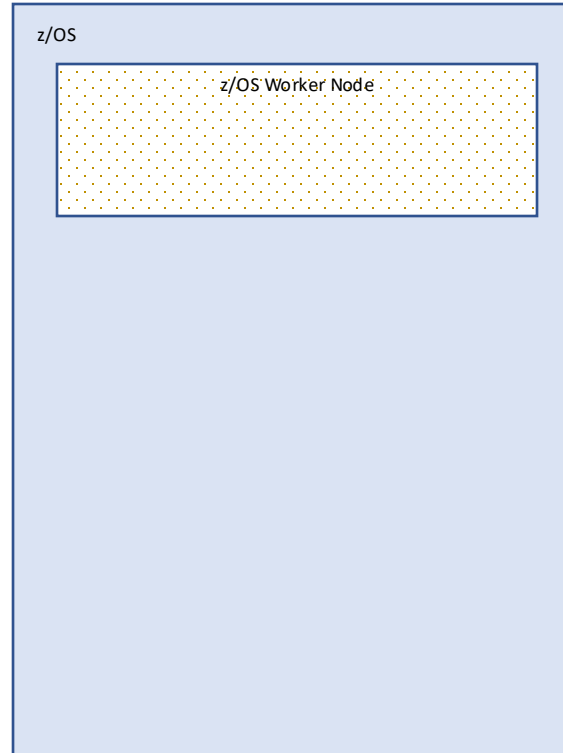
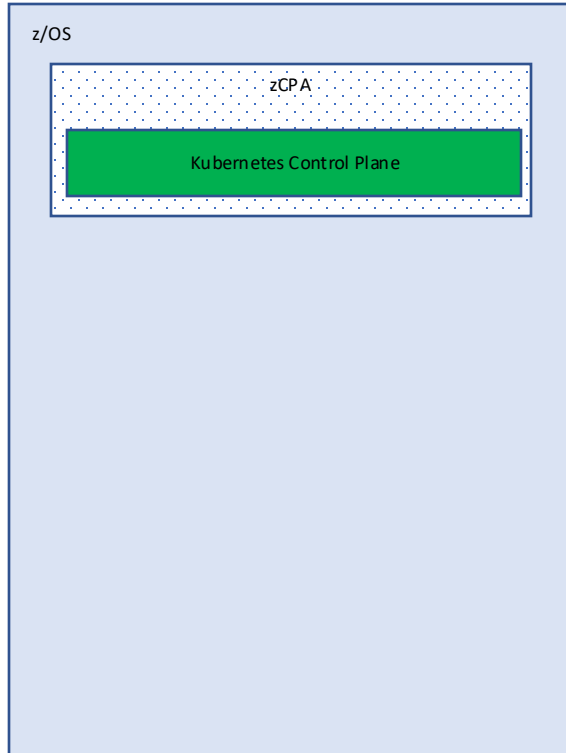
Solution

Create Linux-based appliance that serves as a control plane node



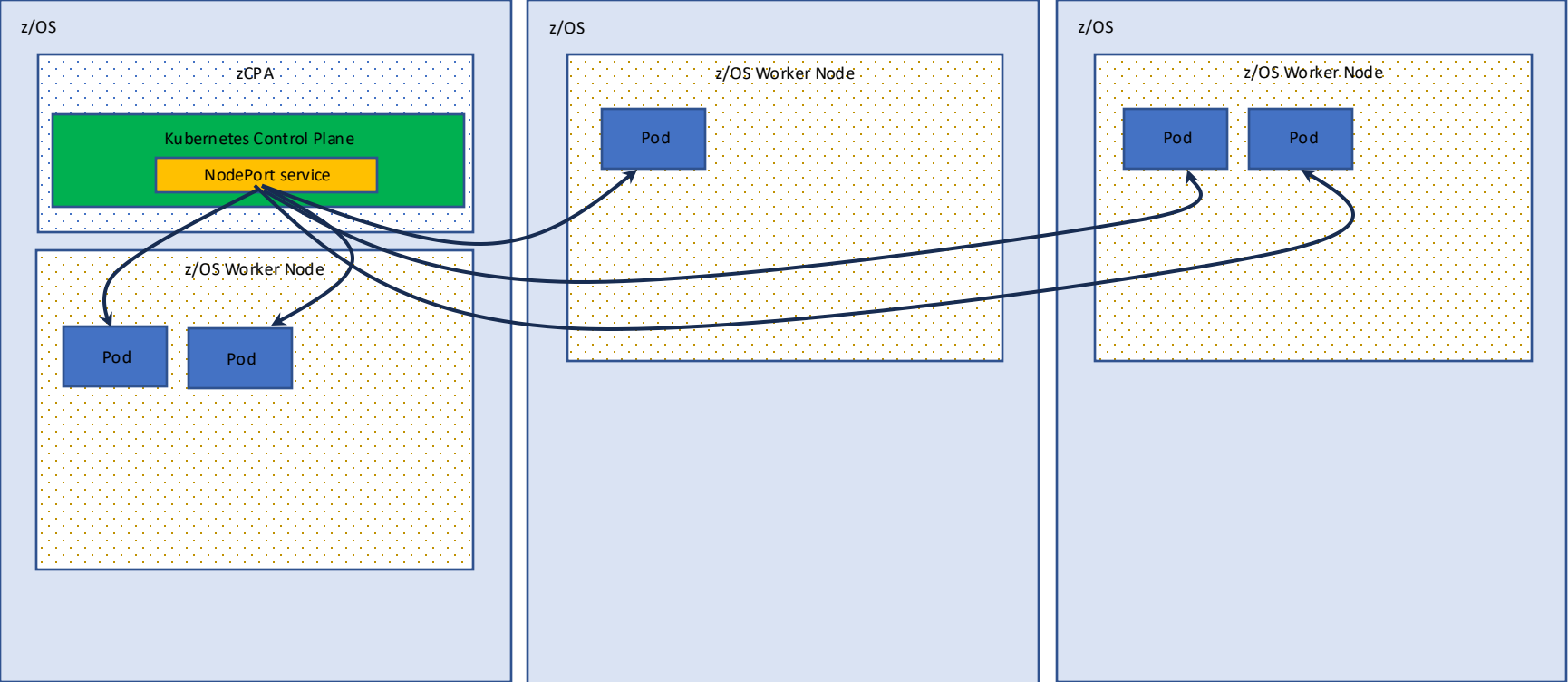
Solution...

Possible configurations



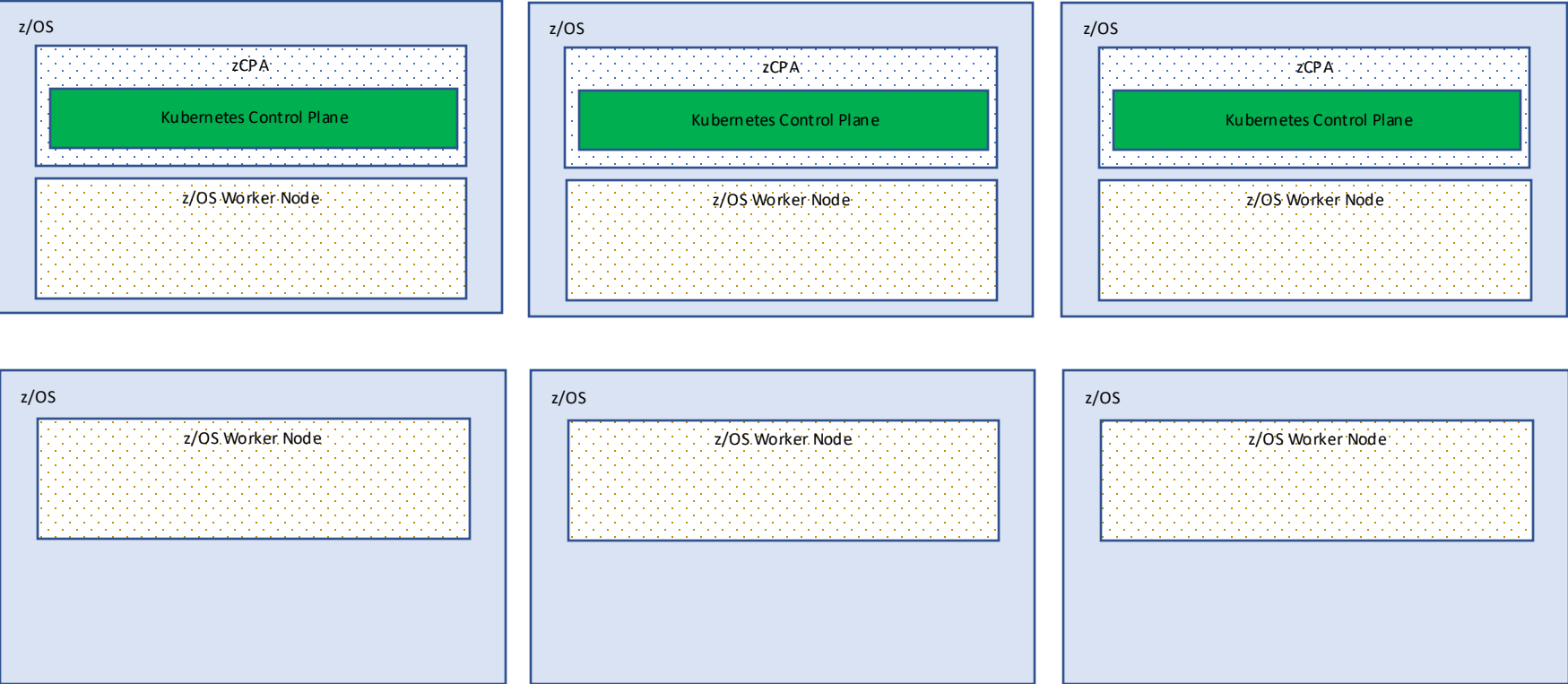
Solution...

NodePort load balancing



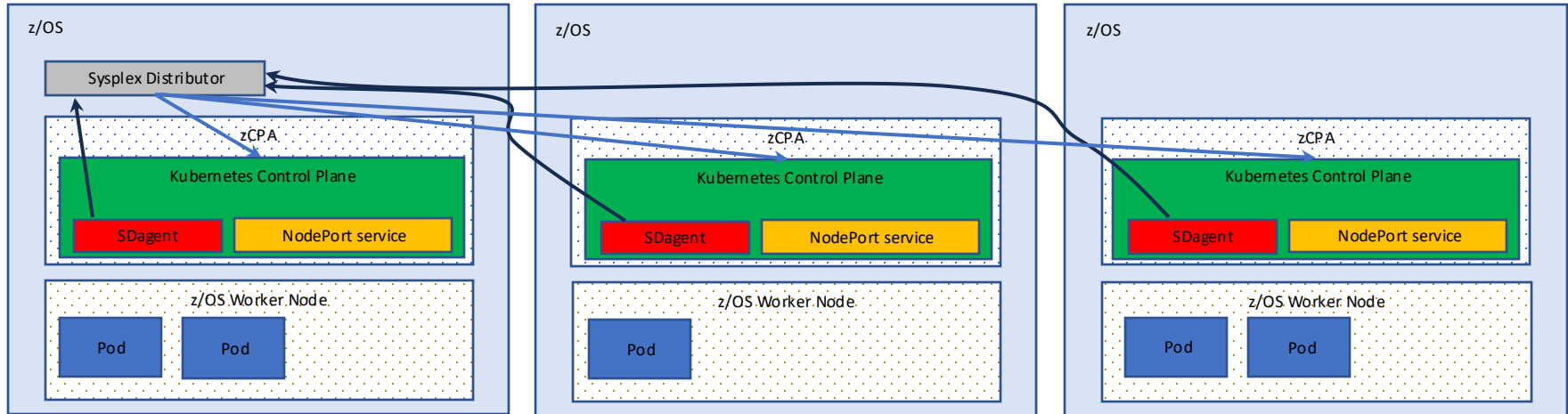
Solution...

High Availability configuration



Benefits

Leverage Sysplex Distributor for High Availability load balancing



zCPA workflows

zCPA Provisioning workflow

Workflows Workflows Settings | Help

Workflows > Provision a Control Plane Appliance (zCPA) Settings | Help

Provision a Control Plane Appliance (zCPA) Notes | History

Workflow Details

Workflow Steps

Actions Search All step content ?

No filter applied

State Filter	No. Filter	Title Filter	CalledWorkflow Filter	Automated Filter	Use RunAsUser ID Filter	Owner Filter	Skill Category Filter
<input type="checkbox"/> In Progress	1	Gather and validate z/OS Control Plane Appliance instance properties					
<input type="checkbox"/> Not Ready	2	Process z/OS Control Plane Appliance instance zFS filesystem					
<input type="checkbox"/> Not Ready	3	Create z/OS Control Plane Appliance instance configuration					
<input type="checkbox"/> Not Ready	4	Allocate and load VSAM data sets for the z/OS Control Plane Appliance disk images		Yes		ibmuser	System
<input type="checkbox"/> Not Ready	5	Generate command to start z/OS Control Plane Appliance instance		No		ibmuser	System
<input type="checkbox"/> Not Ready	6	(Optional) Create sample MOUNT command to add z/OS Control Plane Appliance zFS filesystem to BPXPRMxx		No		ibmuser	System

Total: 22 Selected: 0

[Return to Workflows](#) [Refresh](#) Last refresh: Jul 9, 2024, 12:42:00 PM local time (Jul 9, 2024, 4:42:00 PM GMT)

Main Provisioning Steps

Gather configuration information for zCPA being provisioned

- Determines latest installed version

Allocate and mount zCPA-specific zFS

- Holds configuration files and cached input for subsequent workflows

Create configuration files for zCPA being provisioned

Allocate and build VSAM datasets from provided zCPA image files

- Serves as Linux filesystems for zCPA

zCPA Provisioning workflow – network configuration

Workflows

Workflows > Provision a Control Plane Appliance (zCPA) > 1.2. Gather z/OS Control Plane Appliance Instance properties

Settings | Help

Properties for Workflow Step 1.2. Gather z/OS Control Plane Appliance instance properties

General Details Dependencies Notes **Perform** Status Input Variables Feedback

Input Variables

- z/OS Control Plane Appliance General Configuration
- z/OS Control Plane Appliance General Data Set Configuration
- z/OS Control Plane Appliance CPU and Memory Configuration
- z/OS Control Plane Appliance Network Configuration**
- z/OS Control Plane Appliance Volume Serial Data Set Configuration
- z/OS Control Plane Appliance SMS Managed Data Set Configuration

Review Instructions

Input Variables - z/OS Control Plane Appliance Network Configuration

Enter the variable values for this input category.

* IPv4 Address: ⓘ - IPv4 address for the zCPA instance (as defined by a VIPARANGE ZCPA TCP/IP profile statement):

Sysplex Distributor IPv4 Address: ⓘ - IPv4 address of the Sysplex Distributor DVIPA for the zCPA instance:

TCP/IP Stack Name: ⓘ - z/OS TCP/IP stack name:

* MTU Size: ⓘ - The MTU to use for network communication with the zCPA instance:

< Back Next > Save Finish Cancel

Close

IPv4 dynamic VIPA assigned from
VIPARANGE ZCPA statement in the
TCP/IP profile

IPv4 distributable dynamic VIPA assigned
from VIPADISTRIBUTE EXTARG
statement in TCP/IP profile

zCPA Provisioning workflow – filesystem configuration

Workflows > Provision a Control Plane Appliance (zCPA) > 1.2. Gather z/OS Control Plane Appliance instance properties

Settings | Help

Properties for Workflow Step 1.2. Gather z/OS Control Plane Appliance instance properties

General Details Dependencies Notes **Perform** Status Input Variables Feedback

Input Variables - z/OS Control Plane Appliance Volume Serial Data Set Configuration

Enter the variable values for this input category.

Temporary Work Volume Serial: ⓘ - The volume for temporary data sets for unpacking zCPA binaries:

Root Volume Serial: ⓘ - The volume for the root filesystem for the zCPA instance:

Config Volume Serial: ⓘ - The volume for the config filesystem for the zCPA instance:

Data Volume Serial: ⓘ - The volume for the data filesystem for the zCPA instance:

Dlog Volume Serial: ⓘ - The volume for the dlog filesystem for the zCPA instance:

zFS Filesystem Volume Serial: ⓘ - The volume for the zFS filesystem for the zCPA instance:

< Back Next > Save Finish Cancel

Close

Temporary volume to recombine and uncompress zCPA filesystems

Volume(s) to store VSAM files representing the zCPA mounted filesystems

Volume to allocate zFS for the zCPA instance

- Needs to be large enough to hold any zCPA dumps along with configuration files

SMS-managed storage can be used

Starting the zCPA

```
GLZB025I zCX instance ZCPA1: Initialization has started. Code date 06/14/24.
```

```
:
```

```
EZD1204I DYNAMIC VIPA 9.67.170.243 WAS CREATED USING IOCTL BY ZCPA1 ON TCPIP  
EZD0009I CONNECTION TO 9.67.170.243 ACTIVE FOR INTERFACE EZAZCX
```

```
:
```

```
GLZB022I zCX instance ZCPA1 version information
```

```
Bootloader:          HZDC7C0          oa66001
```

```
3.7.3                2.4.0
```

```
Current Appliance:  HCZ1110          OA66262
```

```
4.4.3                1.29.5
```

```
20240630T205536Z
```

```
Available Appliance: HCZ1110          OA66262
```

```
4.4.3                1.29.5
```

```
20240630T205536Z
```

```
Virtualization Layer: HBB77D0    UJ95425    06/14/24
```

```
Started on 2024/07/02 10:53:25
```

```
Workflows Performed:
```

```
Provision:          1.0.1      HCZ1110    2024/07/01 16:35
```

```
Reconfigure:       N/A        N/A        N/A
```

```
Upgrade:           N/A        N/A        N/A
```

```
Add Data Disks:   N/A        N/A        N/A
```

```
:
```

```
GLZB027I zCX instance ZCPA1: IPLing guest.
```

```
GLZM012I zCX services for instance ZCPA1 are available.
```

Dynamic VIPA
created from
VIPARANGE ZCPA
TCP/IP profile
statement

APAR level and
version of the zCPA
instance

Latest installed APAR
level and version for
the zCPA

Only provisioning
workflow has been
performed on the
zCPA instance

zCPA is now ready to
start control plane
node

Thank you

Redelf Janßen

Contact

E-Mail redelf.janssen@de.ibm.com

Phone +49-171-5538587

