Container Technologies

**Overview:** This document will be used to record container technology system and container orchestration system designs according to the most commonly used open-source containerization frameworks. Brief descriptions that summarize their use-cases, technical features, and integration will be provided as will their strengths and weaknesses. Using critical and soft requirements itemized in the project requirements document, tables will be assembled in a spreadsheet for each system category in the stretch and smart product designs for the purpose of performing a Feasibility Analysis to determine the most suitable designs for each system to down select into the detailed design/prototype-testing phase.

**Containerization Technology System**

**1) Preliminary Design #1: Docker**

**Definition:** Docker is containerization tool that enables the development of containers, or, standard units of software that package up all code and their dependencies so that an application can run quickly and reliably from one computing environment to another. Docker container images are lightweight, standalone, executable packages of software that include everything that is needed to run an application: code, runtime, system tools, system libraries and settings. A container image becomes a container at runtime, and in the specific case of Docker, the images become containers once they run on their specialized runtime software.

**Technical Design of Docker Runtime Environment**

A picture containing text, screenshot, font, number

Description automatically generated

**Strengths**

1) They use minimal system resources

2) Cost-Effective

3) Offer Instant Portability

4) Easy Cloud Deployment

5) CI/CD Support

6) Permit Fast Delivery Cycles

**Weaknesses**

1) Don't support full-system container operations

2) Engine only supports Docker container format

3) Poor monitoring/benchmarking capabilities

4) Difficult persistent data storage

5) Security Vulnerabilities

6) Slow Application Performance

7) Restrictions with GUI's

8) Data Volume Limitations

9) Restrictions with multiple kernels and OS

**2) Preliminary Design #2: Cloud Foundry**

**Definition:** Cloud Foundry is an open-source platform as a service containerization solution. It is a cloud-native platform that can be used to deploy apps to different locations including personal computing infrastructure, commercial computing infrastructure like AWS and others. The platform can run any cloud-native application regardless of what programming language its written in meaning that developers aren't tied into a single framework or set of application services.

**Technical Design of Cloud Foundry Runtime Environment**

**A picture containing text, screenshot, font, design

Description automatically generated**

**Strengths**

1) Open-Source with large international knowledge base

2) Support for many different programming languages

3) Multi-Vendor Support

4) Application Portability

5) Deployed Application Security

6) Ease of Application Deployment

7) Supports Access with SSH

8) Virtual Machine, Disk Space, and Memory Scaling

9) Software Vulnerability Management

**Weaknesses**

1) No support for stateful containers

2) No Persistence of Logs

3) Not Enough Focus on Operational Processes

4) Complexity of Configuring and Monitoring Processes

5) Need to Follow Twelve-Factor App Standards for Development

**3) Preliminary Design #3: CoreOS rkt**

**Definition:** CoreOS rkt is an application container engine developed for modern- production cloud-native environments. It features a pod-native approach, a pluggable execution environment, and a well-defined surface area that makes it ideal for integration with other systems. It is an open alternative to Docker with open operability using an open-source container format. It is focused on container infrastructure management space**.**

**Technical Design of CoreOS rkt Runtime Environment**

A screenshot of a computer

Description automatically generated with medium confidence

**Strengths**

1) Comprehensive Ecosystem

2) Security-Centric

3) Easy to Use

4) Great Documentation

5) Open-Source

6) Regular Updates and Releases

7) API Support

8) Adopted by many Companies

**Weaknesses**

1) Limited Third-Party Integrations and Cloud-Hosted Services

2) Supports only Native Linux Environments

3) Image Signing Verification Required

**Container Orchestration System**

**1) Preliminary Design #1: Docker-Compose**

**Definition:** Docker-compose allows users to describe their application in a docker-compose.yml file including the services, networks, volumes and other aspects of the deployment. Docker-Compose runs docker containers on a single computer, the machine where Docker-Compose is installed and run. It utilizes the Docker daemon to start and stop containers defined in the docker-compose.yml file. It is also capable of creating volumes, networks, port bindings, and other objects available in the Docker API.

**Technical Design of Docker-Compose Runtime Environment**

**A picture containing text, diagram, font, white

Description automatically generated**

**Strengths**

1) Great for Single-Machine Deployments

2) Optimized for Local Development Environments

3) Provide easy and quick configuration

4) Provides security by isolating container operations

**Weaknesses**

1) Time-Consuming to Manually Install and Update docker-compose on machines

2) Health-Checks and lifecycle management tools are not available

3) Containers cannot be replaced without system downtime

4) No access to native package installation managers

5) Changes to containers do not persist after Docker daemon restarts

**2) Preliminary Design #2: Docker-Swarm**

**Definition:** Docker-Swarm provides a container orchestration system to deploy applications across multiple machines running Docker daemon. It works by creating a cluster (also known as a 'swarm') of computers and coordinating the starting and stopping of containers across the cluster. Docker Swarm runs docker containers across a cluster of machines each running docker. It coordinates container execution across the cluster, and, has a similar API to Docker-Compose in that it is capable of creating networks spa

nning multiple computers as well as port-bindings, volumes, and other services.

**Technical Design of Docker-Swarm Runtime Environment**

**A picture containing text, diagram, screenshot, design

Description automatically generated**

**Strengths**

1) Straightforward Installation and Set-Up

2) Lightweight and Easy to Operate for Small Scale Applications

3) Provides automated load balancing

4) Compatible with Docker CLI

5) Can supplement usage of existing docker tools including Docker Compose

6) Configuration changes are not required if system already runs in docker

**Weaknesses**

1) Limited Functionality through dependency on Docker API

2) Has limited fault tolerance

3) Minor community following and support

4) Services must be scaled manually

**3) Preliminary Design #3: Kubernetes**

**Definition:** Kubernetes is a container orchestration system supporting Docker as well as other container runtimes that conform to the Container Runtime Interface (CRI) such as containerd and cri-o. While Kubernetes focuses entirely on containerized applications and is not as similar to Docker Compose as Docker Swarm, it provides a number of powerful features for modern, distributed systems management. Additionally, it is available as a service on a large number of commercial cloud providers. Also, TACC provides multiple Kubernetes clusters in support of various research projects.

**Technical Design of Kubernetes Runtime Environment**

**A diagram of a server

Description automatically generated with low confidence**

**Strengths**

1) Supports running containers across cluster of machines

2) Conforms to Docker Container Runtime Interface (CRI)

3) Provides powerful features for managing distributed applications

4) Available as a service on TACC and commercial cloud providers

5) Open-Source

6) Strong Community

7) Reliable Built-in Container Lifecycle Management

8) Supports Efficient Load Management Across Containers

9) Minimal Performance Overhead

10) Excellent Portability

11) Computationally-Cost-Effective

**Weaknesses**

1) Steep Learning Curve

2) Optimized for more Complicated Applications and Distributed Systems

3) Used mostly outside of Local Development Environments

4) Debugging, Troubleshooting, and Integration require Expertise

5) Scope of automation can put many developers out of jobs

6) Too much startup overhead for small projects