SANDIA REPORT
SAND2020-6250
Printed March 2020

An Approach to DevOps and Microservices

Brandon T. Klein; Gerald J. Giese; Jayson C. Lane; John G. Miner; John J. Jones; Otoniel Venezuela

This report presents work performed in 2017 and is a reissue of the end-of-year report SAND2018-0240, *A Proof-of-Concept Approach to DevOps and Microservices*, *January 2018*, modified for unlimited release.

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

ABSTRACT

This report describes the approach, investigation, and prototyping efforts to create an API management solution and automated deployment pipeline for REST-based microservices. Additionally, it reviews: the microservices created for testing purposes; our experiences and results in using an open-source API management tool; and future plans.

ACKNOWLEDGEMENTS

We would like to acknowledge our student intern team for their valuable contributions in developing a proof-of-concept and exploring the technology stack we chose to evaluate. Their work was invaluable in this effort. Our intern team consisted of:

Aaron Comen

Mark Wesley Harris

Nicholas Tiner

Quincy Lee Conduff

We would also like to thank our Sandia collaborators for their technical expertise and resources in helping us overcome roadblocks and obstacles. Their critical contributions are highly appreciated. Our Sandia collaborators included:

Brandon Showers

Christopher Nebergall

Jeffrey Forster

Jeremy Dencklau

Jeremy Plake

Joshua Barron

Josiah England

Tom Cleal

CONTENTS

LIST OF FIGURES

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
API	Application Programming Interface. This defines the operations and data used to communicate with a piece of software (e.g. an application, microservice, etc.)
Agile	Refers to a group of project management methodologies based on iterative development and collaboration between self-organizing and cross-functional teams.
CaaS	Containers-as-a-Service. An abstraction service framework designed to orchestrate, manage, and deploy container based virtual systems disparately or cohesively.
CD	Continuous Delivery. A software development practice where members of a team produce their work frequently to deliver stable versions at any moment.
CI	Continuous Integration. A software development practice in which members of a team integrate their work frequently, and the integrations trigger verification by automated build and test processes.
COTS	Commercial Off-The-Shelf. Refers to software sold by vendors as ready to use (with some configuration), as opposed to bespoke software developed locally by staff.
CRUD	Create, Retrieve, Update, Delete. These are atomic transactions that can be taken on data, as in documents or data in a database.
Container (software)	A modular, portable, and encapsulated executable software package containing all the proper files and configurations to work properly and consistently regardless of the underlying infrastructure.
Docker	A platform for software engineers and sysadmins to develop, ship, and run applications (https://docs.docker.com/glossary/?term=Docker)
Dockerfile	A text document that contains all the commands needed to execute and build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile. (https://docs.docker.com/glossary/?term=Dockerfile)
Docker EE	Docker Enterprise is a standards-based container platform for development and delivery of modern applications. Docker Enterprise is designed for application developers and IT teams who build, share, and run business-critical applications at scale in production. (https://docs.docker.com/ee/)
Docker Datacenter	Docker Datacenter is a complete, integrated, enterprise solution for container deployment and management, tailored for production environments with a strong emphasis in availability, multi-tenancy and security. (https://hub.docker.com/bundles/docker-datacenter)

Abbreviation	Definition
Docker DTR	Docker Trusted Registry (DTR) is the enterprise-grade image storage solution from Docker. You install it behind your firewall so that you can securely store and manage the Docker images you use in your applications. (https://docs.mirantis.com/docker-enterprise/v3.0/dockeree-products/dtr.html)
Docker Engine	Daemon process running on the host which manages images and containers. (https://docs.docker.com/glossary/?term=Docker)
Docker Hub	A cloud-based registry service which allows access to code repositories, Docker images, and links to Docker Cloud. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline. (https://docs.docker.com/docker-hub/)
Docker Image	The basis of containers, it is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes. (https://docs.docker.com/glossary/?term=image)
Docker Stack	Docker command that allows the deployment of a complete application stack to the Docker swarm (https://docs.docker.com/engine/swarm/stack-deploy/)
Docker Swarm	A standalone native clustering tool for Docker. Docker Swarm pools together several Docker hosts and exposes them as a single virtual Docker host. (https://docs.docker.com/glossary/?term=Docker%20Swarm)
Docker UCP	Docker Universal Control Plane (UCP) is the enterprise-grade cluster management solution from Docker. (https://docs.mirantis.com/docker-enterprise/v3.0/dockeree-products/ucp.html)
Git	An open source version control system for file management
GitLab	An open source, web-based, Git repository platform offering an array of tools and services for software development
JAX-RS	Java API for RESTful Web Services. Part of the Java programming language API specification used in creating web services according to the REST architectural pattern.
Java Spring	A software development framework for the Java programming language.
JSON	JavaScript Object Notation. A structured, human-readable text document.
Kerberos	An authentication protocol developed by Massachusetts Institute of Technology that allows symmetric key cryptography network communication between client and server applications.

Abbreviation	Definition
LDAP	Lightweight Directory Access Protocol. A software protocol for locating organizations, individuals, and other resources on a network, and identifying specific metadata about them, e.g. roles or group membership.
Microservice	An independently deployable, reusable software component that fills a specific function through a defined API.
MSA	Microservices Architecture. An architectural style characterized by independently deployed stateless services (components), typically organized around a business concept and using significant infrastructure automation.
Node.js	A JavaScript runtime build on Google Chrome's V8 JavaScript engine. It is lightweight and efficient with a large ecosystem of open-source software packages available through an online repository with a standard interface.
OAuth	A standard protocol based on token access delegation for authorization over a network.
OpenId	An authentication protocol used over a network standardized by the OpenId Foundation.
REST	Representational State Transfer. An abstraction of the architectural elements within a hypermedia system that ignores the details of component implementation and focuses on the roles of components and their interpretation of significant data elements. Commonly used to refer to web-based APIs using HTTP verbs GET, POST, DELETE, PUT against a URI and exchanging JSON documents or other media.
SAML	Security Assertion Markup Language. An Extensible Markup Language (XML) standard that allows secure network communication between two entities to share authentication and authorization data.
Sandia	Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
Scrum	An Agile framework for completing complex projects a product owner creates a prioritized wish list called a backlog, and a team takes one or a few items from the backlog to complete over a very short period (weeks). Each period is called a Sprint, which begins with planning and ends with a sprint review (including demonstration) and retrospective. Larger teams may have a Scrum master to help with focus and organization.
SDLC	Software Development Life Cycle. Describes the full-lifecycle process of planning, creating, testing, and deploying an information system.

Abbreviation	Definition
Swagger	Part of the OpenAPI Specification, it defines a standard, language-agnostic interface to RESTful APIs allowing both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. A Swagger definition can be used to generate code, documentation, tests, and more.
WSO2	WSO2 is a software vendor specializing in API Management, Integration, Identify & Access Management, Analytics, and Internet of Things (IoT). Many of WSO2's products are available as open source software.
XP	Extreme Programming. A software development methodology intended to improve software quality and responsiveness to changing customer requirements. Most known for the concept of "pair programming" where two software engineers share a computer, one typing and one peer reviewing "on the fly".

1. INTRODUCTION

This paper provides a narrative to document a proof-of-concept approach to DevOps and Microservices that Sandia National Laboratories undertook in FY17.

The selection process for these architectural decisions was derived through the explicit knowledge of an increased return on investment for the business with accelerated time to market and time to value for Sandia's enterprise information systems.

This will describe a subset of Sandia's enterprise information systems vision, the methods employed to make progress toward the vision, and the end state as the fiscal year closed. This serves two purposes: 1) to state Sandia's intentions for planners and architects to consider, and 2) document the end state at the close of FY17, so that a future project team may continue with the approach.

1.1. Vision of the Future

Sandia has formally adopted three key changes to how applications and solutions for the enterprise are to be built: DevOps, Container-as-a-Service (CaaS), and Microservices Architecture (MSA).

In the future, a DevOps-style automation pipeline for microservices-based APIs would be required due to the complexity and quantity of moving parts in an MSA ecosystem. Automation of the pipeline to transition from development to deployment to operation would leverage tools like: Docker containers in a CaaS environment; Swagger/OpenAPI to auto-generate documentation, code and interfaces; automatic testing and monitoring of the APIs; and registration in an API catalog.

The benefits of an automated pipeline are: simplification and acceleration of microservices creation; the ability to find and utilize microservices; and confidence in the qualities of the ecosystem from an architecture standpoint, supporting maintainability, availability, performance, and security.

1.2. DevOps

"DevOps, a clipped compound of "development" and "operations", is a software development and delivery process that emphasizes communication and collaboration between product management, software development, and operations professionals. It supports this by automating and monitoring the process of software integration, testing, deployment, and infrastructure changes by establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably" [1].

Sandia intends to adopt DevOps practices over time, emphasizing automation of complex architecture styles (e.g. MSA) and rapidly provisioned monitoring of components (e.g. Microservices) in a finer-grained manner. Continuous Integration and Continuous Delivery (CI/CD) are the core of DevOps and form a pipeline for moving software from code to operation.

1.3. Microservices Architecture (MSA)

The Microservices Architecture style, or pattern, is used to enable high scalability, availability, and performance of software components. Industry luminary Martin Fowler defines this style in his article "Microservices: a definition of this new architectural term" [2], as having certain characteristics:

- Componentization via Services

- Organized Around Business Capabilities
- Products, not Projects
- Smart Endpoints and Dumb Pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for Failure
- Evolutionary Design

One can deduce from these characteristics that a Microservice is meant to be self-contained and of restricted scope, enabling it to be decoupled to provide robustness and flexibility as it changes.

While a microservice is composed of many small parts, the users see it as a "black box" through its Application Programming Interface (API). Sometimes articles and discussions will use the words API and microservice interchangeably. In this paper, an API is defined solely as the interface, and microservice as a deployable software component.

Sandia intends to adopt the MSA style and patterns for use in enterprise applications and solutions. In addition, Sandia is standardizing primarily on REST [4] and JSON for the APIs.

1.4. Target Architecture

An initial analysis of stakeholder needs yielded the following set of Target Architecture Drivers.

- Provide a common deployment pattern for enterprise applications
- Support DevOps automated deployment pipelines so that development teams can keep up with the needs of business
- Enable sharing of services and functionality across enterprise applications
- Provide constructs for high availability and scalability at little cost to development teams
- Emphasize Continuous Integration/Continuous Testing reducing time to market and errors
- Provide guidance/preferences for Commercial Off-The-Shelf (COTS) solutions which have their own inherent architecture

Research into the MSA style and DevOps automation was then synthesized into an initial target architecture.

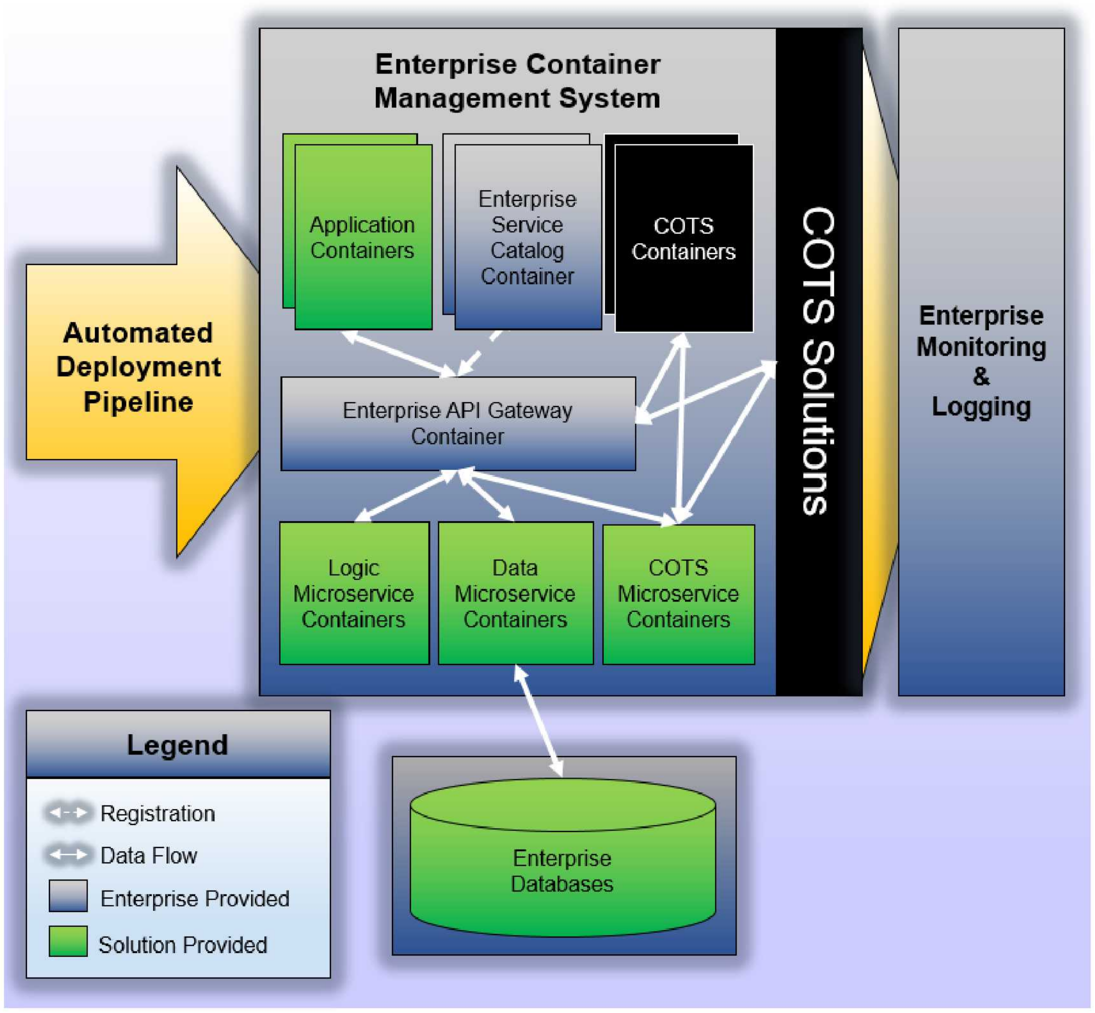
Enterprise Container Management System
Application Containers
Enterprise Service Catalog Container
COTS Containers
COTS Solutions
Enterprise Monitoring & Logging
Automated Deployment Pipeline
Enterprise API Gateway Container
Logic Microservice Containers
Data Microservice Containers
COTS Microservice Containers
Enterprise Databases
Legend
Registration
Data Flow
Enterprise Provided
Solution Provided

Figure 1-1. Target Architecture

Specific standards referenced in the Target Architecture include:

- Microservices shall be designed to be lightweight, independently deployable, stateless, and discoverable
- Microservices shall be RESTful and support the JSON format
- Microservices shall have published Swagger API documentation

1.5. Roadmap

In the spirit of Agile development, and in an attempt to buy down architectural risk, a phased approach was adopted, with each phase expected to be about 6 months in duration. Funding for Phase 1 was established for FY17 and expected to complete at the end of the fiscal year. Funding for Phase 2 and 3 in FY18 is TBD.

1.5.1. Phase 1 – Experimental Research

Phase 1 goals:

- Leverage RESTful microservice design for new applications
- Document microservices with Swagger
- Research, select, and deploy a Service Catalog
- Research and select an API Gateway (for a proof-of-concept)
- Prototype a microservice architecture to be deployed in Docker Data Center
- Promote Continuous Integration/Continuous Testing (fail fast, recover quickly)

This report is a status update for Phase 1.

1.5.2. Phase 2 – Build the Environment

Phase 2 goals:

- Build an institutionally supported Continuous Integration Pipeline for microservices
- Begin doing automated microservice deployments in Docker containers
- Research, select (or develop) and deploy an API Gateway (for production)
- Adopt "Design for Failure" and "Design to Scale" best practices
- Monitor full solution stacks with Enterprise Monitoring
- Deployment/Undeployment of services and applications includes automated registration/de-registration from Enterprise Monitoring

1.5.3. Phase 3 – Operational Rollout

Phase 3 goals:

- All new applications use the Target Architecture and leverage CI/CD practices
- Selection (and purchase) process for COTS products gives preference to technologies, standards and patterns found in the Target Architecture (e.g. RESTful services, containerized deployment)
- COTS services are accessed through the API Gateway

2. PROOF-OF-CONCEPT APPROACH

Sandia established a project in FY17 to explore the MSA style, identify supportive technologies, and test them. A proof-of-concept application and microservices called "Pandamax" was created to learn about and exercise the MSA style, DevOps and Containerization technologies.

The phases were not purely sequential but overlapped and were managed through an Agile process as new information was discovered, discussed, and the product owner made decisions.

2.1. Exploratory Phase

During the exploratory phase, it became clear that complexity and increased numbers of microservices along with their deployments would become an issue if not managed. To mitigate the risks associated with the MSA style, DevOps practices were deemed necessary. The concept of encapsulating microservices in a modular, portable containerized state enabled effective management over the complexity as well.

2.2. Technology Identification Phase

A technology scan for MSA tools and technologies led to the identification of three key technologies types that were determined as needed in our Microservices Target Architecture.

1. Service Catalog – a way for software engineers to find available APIs/services
2. API Gateway – the "front door" for a service or set of services, providing authentication, security, transformation, routing and more.
3. Containerized Deployments – using an Enterprise Container Management System

2.3. Proof-of-Concept Phase

To explore these concepts and technologies, a small team of staff and summer interns tasked to create the Pandamax application and services, and to establish a pipeline to build and deploy them.

The Pandamax application designed a file management utility, which can upload, update, and delete files, as well as return statistics on an uploaded file. It was created to test multiple common web application features and do so in a way that would also test the API Gateway. The microservices support "delay" parameters to simulate various load and performance issues that may be experienced when leveraging a centralized API management solution. The application features and service API essentially mirror each other, with the API Gateway mediating between services to explore the gateway functionality.

This proof-of-concept will help us understand and build the infrastructure services necessary to support a larger ecosystem.

This page left blank

3. EXPLORATORY PHASE

During the Exploratory Phase, it became clear that the desired Enterprise Service Catalog and Enterprise API Gateway fall under a category called API Management. Some vendors have all-in-one offerings, while others take a suite approach and offer tools or platforms for each capability.

3.1. Desired Features for a Microservices Architecture

Three main features were identified to alleviate the complexity of API management and microservice deployment in an MSA: an API service catalog for internal software engineers; an API gateway to control access to microservices; and containerized microservice deployments.

1. Enterprise Service Catalog
    - Provides a user-friendly view of available microservice APIs and information on how to consume them.
    - Swagger, also known as OpenAPI, is a tool for defining microservice APIs and supports automatic generation of API documentation.
2. Enterprise API Gateway
    - Creates a virtual layer in front of microservice APIs, insulating clients from changes at the service layer. Can provide a centralized point for authentication, security, traffic control, logging, transformations, load balancing, analytics and more.
3. Containerized Deployments
    - Docker technology is heavily used in industry and creates a very lightweight and portable container, analogous to a virtual machine, in which to deploy a software component and its dependencies.
    - Docker Swarm provides orchestration of Docker containers, while Docker EE adds a registry, security and management functionality. Docker EE is being evaluated by the Enterprise Cloud Services (ECS) team at Sandia.

3.1.1. Enterprise Service Catalog

Sandia's intended usage for a Service Catalog is as a catalog for software engineers to identify available services and get the information necessary to understand their functionality and begin using them. Some vendors call the Service Catalog a Service Registry or an API Marketplace. Many vendors target this functionality as a business-to-business or business-to-consumer tool for monetization purposes, where one company sells access to its API to another company. Sandia is not currently pursuing this type of functionality as its users are purely internal.

3.1.2. Enterprise API Gateway

Microservices are developed independently, so cross-cutting concerns are dealt with by other layers in the software stack. The API gateway is one of those layers.

Here is a list of common concerns handled by API gateways:

- Authentication

- Application service subscription
- Transport security
- Load-balancing
- Request dispatching (including fault tolerance and service discovery)
    - Including aggregation of multiple requests results
- Dependency resolution
- Transport transformations (protocol and/or format)

3.1.3. Containerized Deployments

During the exploratory phase, several team members interviewed software development teams already using Docker and Docker swarm for deployments. Based on the encouraging results they had experienced, the team researched robust management tools and platforms that would support multi-tenant deployments of Docker containers with more automation and administrator support functionality.

4. TECHNOLOGY IDENTIFICATION PHASE

As a result of the Exploratory Phase, it was determined that the team should focus on selecting an API Management technology that could utilize Sandia's CaaS prototype for the proof-of-concept. This section describes the process and how it was executed.

4.1. API Management Decision Process

The process for determining which API Management technology to prototype with was as follows:

1. Determine Sandia's desired features for an MSA API management tool suite.
2. Quickly identify products that appear at a glance to be candidates, leveraging both open-source and COTS products identified as industry leaders in the Gartner Magic Quadrant for API Management tools.
3. Capture information (website, email, etc.) for each product that show how it meets, or does not meet, the desired features.
4. Create a Decision Analysis and Resolution (DAR) Report. The DAR template used was made in Excel and consisted of 4 tabs:
    - Report – an executive summary outline of the problem, desired features, scoring, and result.
    - Decision Matrix – the evaluation attributes, weights, scores, and results chart
    - Result – notes on how the products achieved (or not) each evaluation attribute
    - Enterprise Architecture – a yes/no 'supports' matrix for higher-level desired attributes of the larger enterprise architecture
5. Hold a short series of meetings to collaboratively fill in the DAR with scores for each product, while discussing the features to share knowledge and insight.
6. Review the results with the team to make a final selection.

4.2. API Management Desired Features

The following features were brainstormed based on needs and research performed in the Exploratory Phase:

- Inexpensive – e.g. cost per 300,000 calls
- Authentication – OAuth 2 and SAML
- Authorization
- Input Validation/Endpoint Protection
- Traffic Control
- Open Source and Support Model both offered
- Scalability
- Availability

- On Premises Deployment Available
- Testability
- Discovery API and Web Interface
- Messaging Support – e.g. WebSockets/JMS/AMQP/STOMP
- Portability/Architecture
- Performance/Throughput – e.g. nominal deployment architecture is 2 sets of 2 nodes
- Logging/Audit/Analytics – e.g. Splunk and ELK
- Open API Support
- Development Support – e.g. discovery via Eclipse and Visual Studios plugins
- Simplicity

4.3. API Management Technologies Evaluated and Selected

The API Management technologies were evaluated based on the criteria listed in section 4.2; WSO2 was selected.

5. PROOF-OF-CONCEPT PHASE

The proof-of-concept was meticulously crafted to model proven methodologies regarding Microservices Architecture, DevOps and Containerization. The initial technologies selected were: WSO2, Docker, Git, GitLab, Jenkins, and internal Sandia cloud resources. The end goal was an orchestrated, managed, and automated CI/CD containerized ecosystem consisting of a tailored API manager configuration and custom engineered microservices to serve as a proof-of-concept for the Enterprise.

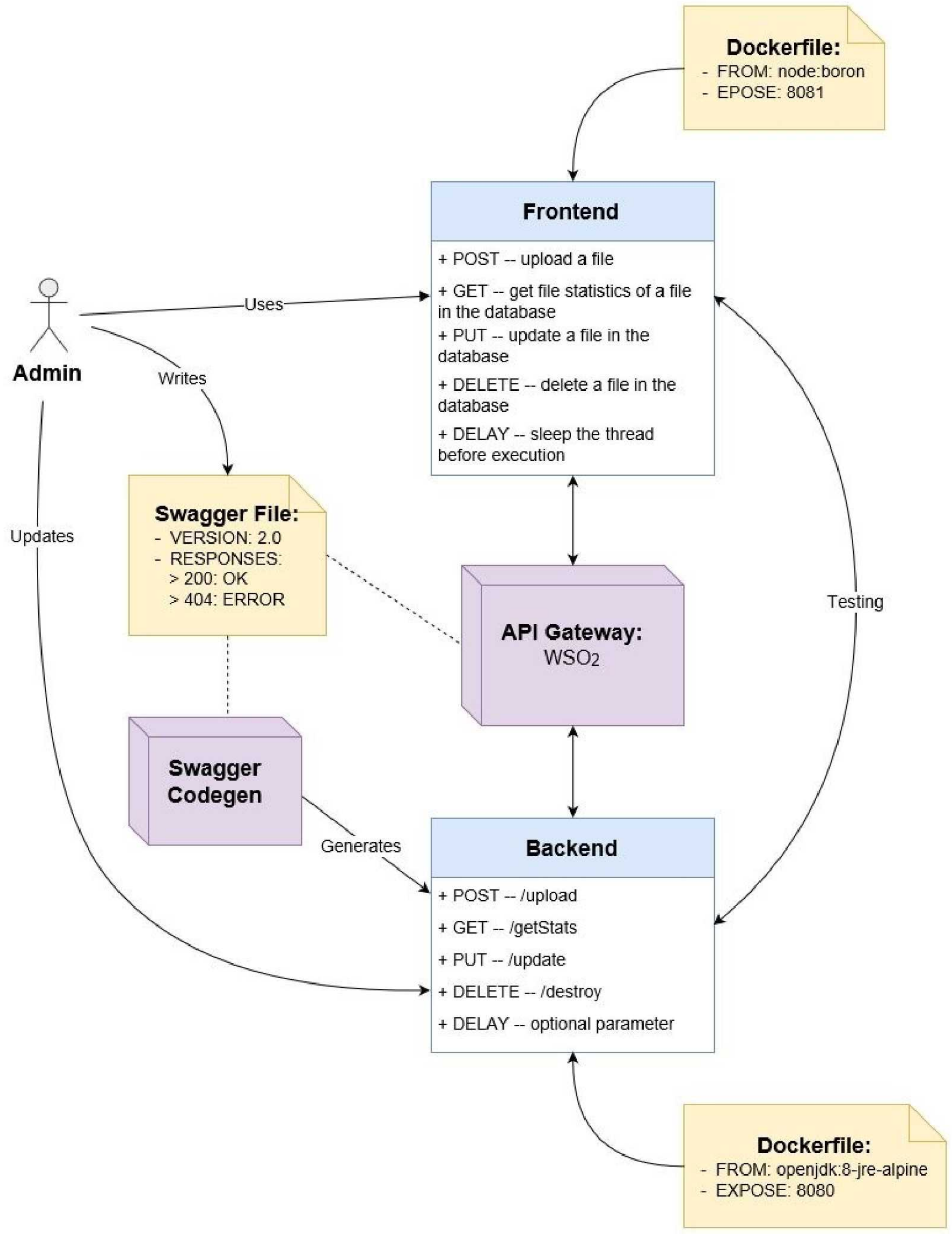

Figure 5-1. Pandamax Architecture Model

5.1. DevOps, Agile, and Interns

The Pandamax team selected the adaptive SDLC approach for the project's framework with incorporation of DevOps and Agile methodologies. Scrum and XP were implemented as the Agile methodologies for the duration of the project.

Additionally, summer internship positions were extended to four college students to partake in the project's engineering endeavor. The interns had little to nil prior experience with containerization, MSA and DevOps; however, the interns were able quickly learn and contribute to the effort.

5.2. Containerization and Cloud Infrastructure

Containerization and cloud computing were technology choices selected by the Pandamax team due to the major engineering and business advantages they provide to production: accelerated time to market and time to value. Docker was the selected container technology due to its industry adoption size, market share, and capabilities. Internal Sandia cloud resources offered by the ECS team were selected for the cloud infrastructure due to the perceived affordance of in-house cloud.

The Docker capabilities leveraged for the project included: Docker Engine, Docker Swarm, Docker Stack, Dockerfile, and Docker Hub. The Pandamax project leveraged Docker resources from the ECS team as well. The ECS team offered supported Enterprise Docker Swarm on RHEL 7 virtual machines. Thus, four virtual instances were created and clustered via Docker Swarm. The Docker Swarm served as a deployment location for the project. Additionally, the Pandamax team partook in the initial offering of the ECS team's CaaS prototype leveraging Docker EE.

5.3. Pandamax Application and Service

Microservices were needed to test the WSO2 API Manager as well as the deployment methodology. Ergo, the development team decided to create two custom individually containerized REST based microservices. The selected Microservices were trivial in nature; however, their functionality fulfilled an archetypical business need – CRUD operations.

The first microservice was a frontend service for Pandamax written in Node.js and HTML with GET, POST, PUT, DELETE functionality. The functions of the frontend service had the option to be delayed as well. The second microservice was a backend service for Pandamax written in Java separately with Java Spring and JAX-RS for testing purposes; Swagger-Codegen auto generated server code. The backend service accepted files (ingress) and shared (egress) information associated with them; the files were stored locally on the backend service container.

The frontend service connected to the backend service and vice versa through WSO2. An access token was used to communicate with an API or a logical collection of associated APIs within the WSO2 API Manger.

Figure 5-2 depicts the Pandamax application as tested.

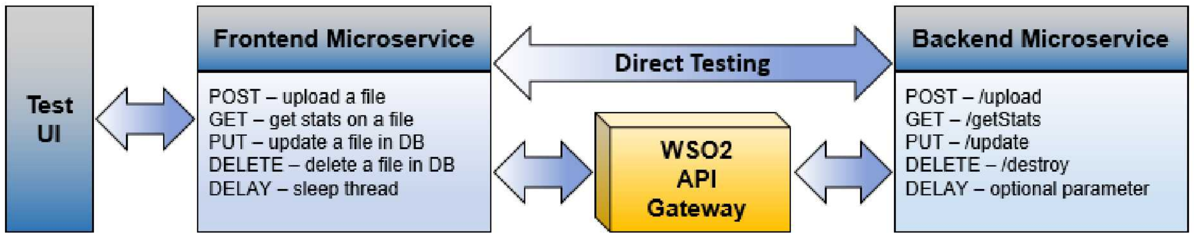

Figure 5-2. Pandamax Application and Services

5.4. CI/CD with GitLab and Jenkins

The CI/CD pipeline for Pandamax was implemented via GitLab CI and Jenkins, depicted in Figure 5-3. All code associated to the project was managed on the internal Enterprise GitLab. The Enterprise GitLab managed the CI via GitLab's Webhook event triggers. The GitLab CI built and tested images triggered by a code push to the master branch of the project's code repository on the Enterprise GitLab. The CI pipeline created two different versions of the Docker images: 'test' and 'production' during the Build stage. The Test stage ran unit tests against the 'test' images. Meanwhile, the Deployment stage deployed 'production' images to the GitLab container registry.

The Jenkins application performed the CD as a Docker container residing in the project's Docker Swarm cluster. Jenkins pulled the project's code from the GitLab repository contingently based on confirmation from GitLab CI that the repository indeed changed. Jenkins then deployed a Docker Stack to the project's Docker Swarm cluster. For deployment to the ECS Docker EE offering, Jenkins did not interact directly with the deployment; instead, it built and pushed all images to the DTR needed for the Docker Stack. The ECS Docker EE offering used an intermediate process to launch the Docker Stack and updated any pre-existing services after the images were pushed to the DTR.

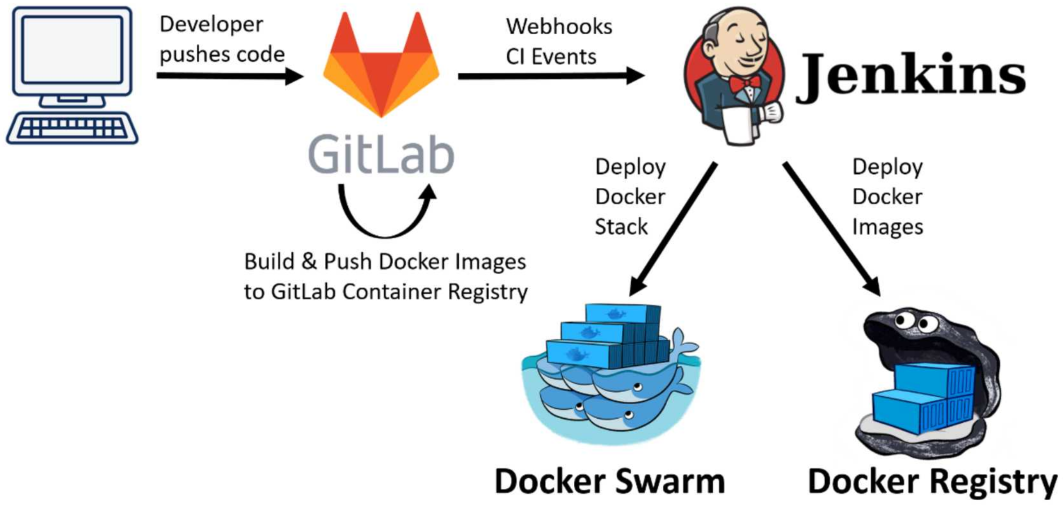

Figure 5-3. Pandamax CI/CD Pipeline

5.5. WSO2 API Manager

The WSO2 API Manager was selected as the API Manager for the Pandamax proof-of-concept. WSO2 released its source code on GitHub and produced scripted methodologies to containerize their API Manager into Docker images. Due to WSO2's portable and modular design, integration with the Pandamax project's Docker Stack required minimal effort.

5.5.1. API Catalog

The WSO2 API Manager – API Store served as the API catalog. The API Store hosted published APIs and allowed for API discoverability, subscription, and registration of applications. WSO2's API Store leverages the Swagger API definitions to provide detailed documentation on the API's utilization. This can be leveraged by software engineers as a one-stop-shop to search and discover APIs.

5.5.2. API Management

APIs were managed through the WSO2 API Manager – API Publisher. The API Publisher enabled the creation, development, publication, examination, documentation, and management of APIs. API Publisher supports Swagger integration for API design and documentation.

5.5.3. Authentication and Authorization

To test the viability of the API Manager for the Enterprise, authentication and authorization were deemed appropriate configuration requirements. Thus, the team ventured integrating the API Manager with the Sandia Enterprise systems via LDAP/Kerberos, SAML, OAuth, and OpenId.

6. END STATE

This section presents the results for each objective for Phase 1 of the Target Architecture at the close of FY17:

6.1. Leveraging RESTful microservice design for new applications

This objective was completed through posting and communicating the Target Architecture, and through feedback given in mandatory reviews in the Director's Architecture Working Group, a part of the Solutions Deployment Lifecycle (SDL).

6.2. Documenting microservices with Swagger

Prototyping this technology was trivial, and the Pandamax application's use of Swagger verified the utility and simplicity of Swagger. This objective was further completed through posting and communicating the Target Architecture, and through feedback given in mandatory reviews in the Director's Architecture Working Group, a part of the SDL.

6.3. Research, select, and deploy a Service Catalog

This objective was only partially met. Deployment and testing of the service catalog was successful; however, due to time and human resource constraints, population of the API Store with existing Enterprise microservices was incomplete.

6.4. Research and select an API Gateway

This objective was completed and WSO2's API Management toolset was selected and prototyped. At the time of this writing, software engineers are smoothing out the security model, but the core functionality has been verified and tested using the Pandamax application.

6.5. Prototype a microservice architecture to be deployed in Docker Data Center

This objective was completed. The project successfully tested the management of Docker images through the ECS DTR and the deployment of Docker Stacks to the ECS Docker UCP. There were configuration and methodologies learned from the experience which were beneficial to both the Pandamax team as well as the ECS team.

6.6. Promote Continuous Integration/Continuous Testing (fail fast, recover quickly)

This objective was completed through posting and communicating the Target Architecture, and through feedback given in mandatory reviews in the Director's Architecture Working Group, a part of the SDL. In addition, a prototype CI/CD pipeline was created to deploy WSO2's API Management solution and the Pandamax application.

This page left blank

7. NEXT STEPS

As FY17 ended, some minor work was left unfinished, but this work falls neatly into the initial work needed for Phase 2.

Several tasks are needed for the transition from Phase 1 to Phase 2:

- Complete integration of WSO2's API Management solution with Sandia's Shibboleth single-sign-on. There are a few minor outstanding issues documented in the Pandamax's GitLab project repository.
- Complete integration of WSO2's API Management solution with Sandia's OpenID Connect authentication service. There are a few minor outstanding issues documented in the Pandamax's GitLab project repository.
- Deploy WSO2's "Pattern 3" API Management architecture to prototype high availability and scalability of the API Manager.
- Perform load testing against WSO2's "Pattern 3" API Management architecture.
- Populate the API service catalog with several microservices so they are visible in the deployed WSO2 API Manager – API Store; verify capabilities and usability for Sandia software engineers.
- Apply the lessons learned from implementing the WSO2 API Gateway proof-of-concept and revisit the API Management DAR, including the addition of PESTLE and SWOT analyses.

This page left blank

REFERENCES

[1] Wikipedia contributors, "DevOps," *Wikipedia, The Free Encyclopedia.* Accessed September 5, 2017 https://en.wikipedia.org/w/index.php?title=DevOps&oldid=797774921

[2] Fowler, Martin. "Microservices." Martinfowler.com. Accessed September 05, 2017. https://martinfowler.com/articles/microservices.html.

[3] Olliffe, Gary. "Guidance Framework for Evaluating API Management Solutions." Technology Research. November 16, 2015. Accessed August 25, 2017. https://www.gartner.com/doc/3168517/guidance-framework-evaluating-api-management.

[4] Fielding, Roy Thomas. “Architectural Styles and the Design of Network-based Software Architectures.” Doctoral dissertation, University of California, Irvine, 2000.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	01977	sanddocs@sandia.gov

This page left blank