Job Aid: Common Concepts in microServices

Continuous Deployment Platform

Dewayne Hafenstein

Principal Technical Architect

Contents

Overview

microServices has many technologies, approaches, conventions, and terminology.

It can be confusing to understand what different technology to use in different cases, or even what the different technologies offer.

You may have heard most if not all of these terms. But how do they fit together? What are they and how are they used?

How do microServices use these? And, what, exactly, are “microServices”?

This job aid takes a short look at some of the common concepts including:

- Domain
- Domain-Driven Design
- Service
- SOA
- Container
- REST*ful* API
- Virtualization
- Docker
- Kubernetes
- CI/CD
- CDP
- microServices

What is a Domain?

A domain is simply a collection of things for the purpose of a common understanding or discussion.

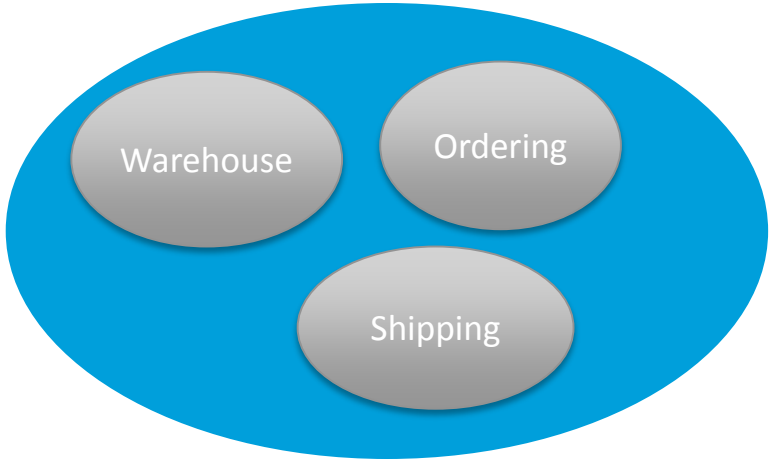
The word “domain” means a specified sphere, or collection, of activity or knowledge, for example:

- The domain of positive integers is the collection of integer numbers from 0 thru infinity.
- The domain of a business is the collection of all knowledge and business operations that are performed to support that business.

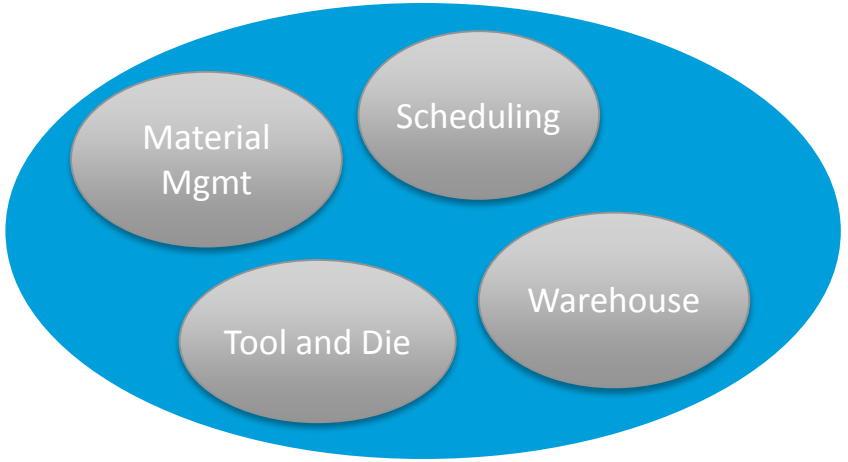
What is a Domain, continued

A business domain is the collection of all activities that are performed within some boundary of the business.

- For example, a retail sales company probably performs warehousing, receiving, shipping, order management, billing, and other business activities. These activities and the business knowledge form the business domain.
- A retail business would likely have a different business domain than a manufacturing concern.
- Traditionally, reuse of software components has been low because of poor domain reusability.



A Retail Sales Domain



A Manufacturing Domain

What is a Domain, continued

Business Design Analysis in microServices

The important thing in the analysis of a business for design of microServices is to realize that we are concerning ourselves with the business operations.

- We are modeling things like placing an order, not logging a transaction.
- The implementation details are not important, and tend to confuse the analysis.
- Keep the analysis in terms of what operations need to happen, the flow of information, and the business activities.
- Do not focus on the low-level implementation details yet.
- Those will be addressed after analysis and design, and the determination of what services are needed.
- You want to maximize the reusability of the service. This means that the service needs to be defined in the terms and processes of the business. If it is polluted with implementation specializations or poorly modeled, the reusability will suffer.

What is a Domain, continued

A domain is a collection of knowledge—so it has its own terminology, based on the business.

- The problem faced by most analysts and developers in domain-driven development is that the subject matter experts (SMEs) don't speak the same language as you do. SMEs understand the domain. Developers and analysts typically do not!
- It is not possible or desirable to teach the SMEs to program, nor is it desirable or practical to teach the analysts and developers the domain.

There is a Communications Gap—and if there is any misunderstanding, there is a risk that the product will be incorrect, or worse unusable.

What is a Domain, continued

How do we communicate?

- Teach everyone the domain?

  This will take too long and cost too much, and the risk is high.

- Teach the SMEs how to design and develop software?

  This will also take too long and cost too much, assuming that the SMEs would even be willing to do this.

- Design an abstraction that uses different terminology and models that are understood by development?

  This risks development of a product that does not meet the needs of the business. It also introduces all sorts of risk and error because the SMEs and analysts do not have a common understanding of the analysis.

- **Find a common representation and terminology that simplifies the domain enough that it is still meaningful but is accurate, and that everyone can understand?**

  This is the essence of domain-driven design. We need some common understanding of the business domain and the problem being addressed so that the analysts, developers, and SMEs can all understand each other.

  This common understanding is essential so that any analysis can be verified by the SMEs, and the SMEs can provide input to the analysts.

What is Domain-Driven Design?

Domain-Driven Design (DDD) is based on a simplified model and common vocabulary developed in conjunction with the SMEs and analysts/developers. Key features include:

- Analyzes the business activities from the point of view of the domain.
- Establishes a common “language” and vocabulary, based on the domain.
- Establishes a common understanding so that the SMEs, Analysts, and Developers can all have a common understanding.
- Constructs a simplified model of the domain for use in analysis and development.
- Manages different perspectives of the domain.

What is Domain-Driven Design, continued

What is a model in DDD?

Models are simply mechanisms for the description and understanding of business processes. Models allow for easier understanding of complex subjects because they allow us to focus only on the salient details. Typically, these models are UML diagrams.

- The model should be simplified enough so that it is still accurate, everyone can understand it, and it pertains to the problem being addressed.
- Do not complicate the model with details that do not pertain to the problem.
- Develop, maintain, and use a common vocabulary (terminology and meaning) and use it all the way through the development process; from analysis all the way through coding and documentation.
- Use terms that are meaningful to the SMEs, do not “make up” new terms.

What is Service Oriented Architecture (SOA)?

A service is an independent software component that performs some activities on demand. The essence of SOA is to break up the business domain into reusable services and to use those services to create applications. SOA is used in the following ways:

- As a style of software architecture.
- To assemble an application by integration of distributed, separately developed and maintained, reusable application components which interact with each other through well-defined interfaces using communications protocols over a network.
- To place emphasis on the services representing business domain functionality.
- To allow the service implementations to be independently developed, maintained, and deployed.
- To allow the service implementations to vary independently over time. Services are considered “black-boxes” to the application using them, and the actual implementation is immaterial.
- To allow services to move, scale up or down, and to be dynamically located through a service registry or service discovery.

What Service Oriented Architecture (SOA), continued

Since all interaction is via their API over a network, services could be implemented in completely different technologies. Whatever is best for the implementation of the service.

- The implementation of those services is not mandated by the architectural style.
- SOA does not mandate any specific language or technology.
- It can be implemented using Java, C#, C++, Node.js, or just about any other technology.

What Service Oriented Architecture (SOA), continued

A microService is a specific way to implement a service. It does not mandate the technologies used, just how they are developed, deployed, executed, and accessed.

- microServices are container-based. microServices are service implementations deployed and managed by an unshared container.
- Containers manage the infrastructure and dependencies. The container provides the infrastructure isolation and hosts the service.
- Each microService instance runs in its own container. Containers host one and only one service per instance (unlike web, EJB, or OSGi containers).

A microService is an independent, reusable, network-accessible, discoverable, location transparent, portable, business domain function.

- This definition does NOT imply or dictate any specific implementation. In fact, there can be any number of implementations and technologies that can be used. There is no restriction that all services need to be implemented in the same technologies either.

What is Service Oriented Architecture (SOA), continued

microServices can be used in an implementation of SOA!

– There are many ways SOA could be implemented. microServices is one implementation pattern or approach.

– Using microServices does not mean it is SOA!

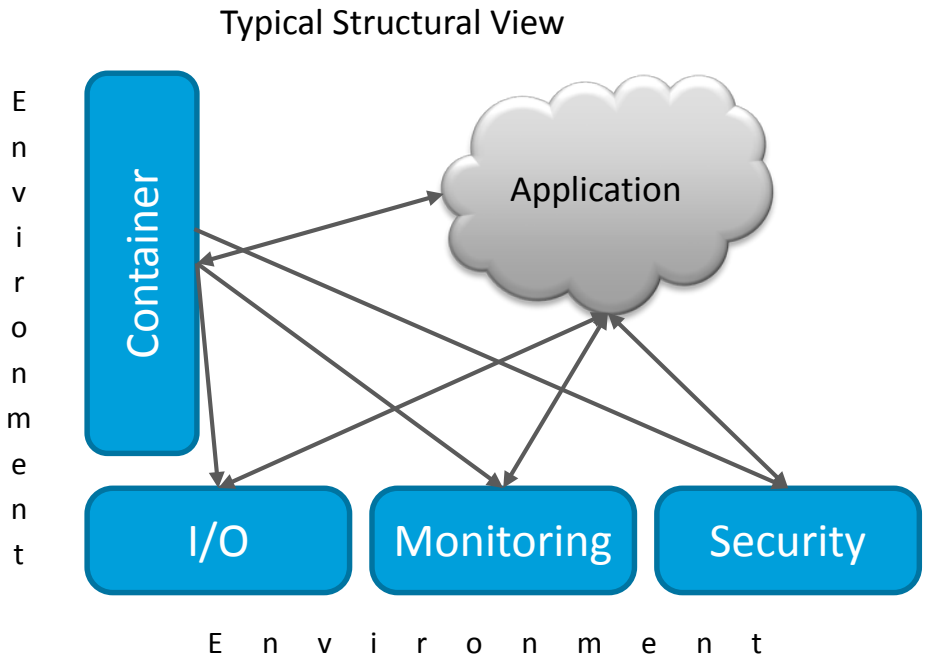
microServices addresses some of the complexity to implementing services oriented architectures!

- microServices address the packaging, deployment, execution, monitoring, security, registration, discovery, and other run-time aspects of services in a standard way.
- microServices do not dictate the technologies used to implement the service. The goal is to separate the infrastructure management from the service implementation. The container handles all the infrastructure management and allows the service to focus on the business. Using standard container technologies allows the service implementations to be simplified.
- Because microServices are a way to package and run a service, and it is possible to create a monolithic “service”, using microServices does not automatically imply SOA. microServices are a packaging and runtime technology that supports the use of services, but does not guarantee good domain-driven design, service API design, or any other aspects of good SOA implementations.

What is a Container?

Containers are software frameworks that host components.

– In the diagram below, the container handles all of the I/O, monitoring, and security overhead (environment) while the application is free to execute business processes. This helps applications be reused over many different types of containers with different environments.



What is a Container?

Container technologies have been around for a long time.

The goals of container-based approaches is to separate the environmental and framework aspects of an application from the business logic. If the application component is separated from the container:

- The application is easier to develop, understand, support, and extend.
- The application is portable across any implementation of the container.
- The application is secure (delegating security to the container).
- The application is helped by automatic scalability and fault tolerance (if supported by the container).

Containers are frameworks that provide a hosted environment for an application component. The component interacts with the container, and the container manages its environment for it. This delegation of infrastructure management to the container makes the application component much simpler.

What is a Container, continued

Containers provide an execution environment for software components

Containers have always attempted to separate the infrastructure and environmental aspects of an application component's lifecycle from the actual application component implementation. This means that a component developer no longer needs to write code to handle security, fault-tolerance, scalability, or many other details about the execution of the code. Instead, when running in a container, the application code can focus on the business logic.

- Containers manage the environment that the application runs in the following ways:
  - They manage input and output.
  - They manage memory.
  - They usually provide security.
  - They usually provide monitoring and management capabilities.
  - They usually manage the lifecycle of the application.
- Generally, containers provide several benefits including:
  - Framework and platform neutrality
  - Portability
  - Consistency
  - Security
  - Simplification
  - Isolation

What is a Container, continued

There are many different container technologies.

A good number of these developed in the java world because of its ease of adaptation to this approach. However, the concept of containerized software is not limited to java, and in fact can be extended to any language or technology if the container provides the equivalent of the operating system “process”. This is essentially what the Rkt and Docker containers do. This allows the application component to be literally any implementation that can run inside a process including scripts, Node.js, C++, Java, or any other technology.

Many container technologies (OSGi, EJB, etc..) are “shared” containers, hosting multiple application components in the same instance.
Some, like Docker and Rkt are unshared containers. In these cases, each instance of a container hosts one, and only one, application component.

The environment created by the container is not shared with any other application component. To host multiple application components, multiple containers are used.

What is a Container, continued

Here are some examples of container technologies.

OSGi

- OSGi (Open Service Gateway Initiative) is a Java framework used to develop and deploy software programs and libraries in a modular fashion.

EJB

- An EJB Web container provides a runtime environment for Web-related software components such as security, servlet management, and transaction processing.

WEBApp

- A Java Web application container

Lxc

- Linux container that allows multiple isolated virtualized Linux systems to be run on a single host.

Rkt

- A container manager for Linux that runs applications in isolation on a host system.

Docker

- A container manager for Linux that runs applications in isolation on a host system.
  Note: *Microsoft has announced that it will support Docker on Windows server in an upcoming release.*

What is a Container, continued

Containers are NOT microServices--Running application components in a container does not make them a microService!

microServices build on the concept of containerized software components, adding much more capabilities than the simple container. Therefore, a microService will be running in a container, but running in a container does not make you a microService!

- Containers are means to run application code in a protected, common, portable, and controlled environment.
- Containers isolate the application code from the environment and infrastructure.
- Use of containers to realize microServices is essential. However, an application running in a container is not necessarily a microService.

What is Docker?

Docker is a container implementation.

The Docker container virtualizes the execution environment presented to the component it is running. The component deployment package contains all of the dependencies the component needs to run. A descriptor file is created by the component developer that describes the needed environment to Docker. The Docker container loads and hosts these dependencies for the component, on whatever system the Docker container may be running. In effect, the Docker container simulates the operating system and execution environment for the component.

- Docker uses application virtualization.
- All of the dependencies of the application are managed by Docker.
- Each container instance is isolated and independent.
- An application is packaged into an “image” that contains the application and all of its dependencies.
- One Docker image runs in one Docker container. The Docker container is not sharable.
- Docker containers run as processes on the host system.
- An instance of a Docker container runs as one process on the host system, and supports one component and its declared dependencies.

Note: Using a Docker container does not make the component a microService! A monolithic, 2-tier, or peer-to-peer, or components of an n-tier architecture can be deployed as Docker container-managed components. A microService uses a container, but using a container does not make you a microService.

What is Kubernetes?

Kubernetes is a platform for managing containers

It has the ability to schedule and run containers on any number of hosts (real or virtual). It monitors the containers and can automatically start new ones in the event of a failure, and scale up or down instances. Key features include the following:

- Automates scaling of container instances.
- Automates recovery of failing container instances.
- Manages the roll-out of new container versions.
- Manages the roll-back of versions.

How does Kubernetes relate to microServices?

- Using Kubernetes to manage your container-managed components does NOT mean your components are microServices.
- microServices will use containers, and they are usually managed by some form of runtime manager like Kubernetes.

Kubernetes

What is a REST*ful* Interface?

API Definition and Usage

An API is the definition of an interface to some software component. It defines the operations that can be requested, the properties or parameters that must be supplied, and the results that can be obtained for each operation.

- As such, the API is like a “contract” in that it guarantees that if a client calls the operation defined, and supplies the proper information in the proper context, that the defined result will be obtained.
- It is a contract to not only the component that is implementing the operations, but also to the clients that are using them.
- Neither side can deviate from the contract without causing the interaction to fail.

There are a lot of cases where software components need to expose an API. A 2-tier web-based application can expose an API to allow the client to interact with it.

- An N-tiered application can expose APIs at various levels. Peer-to-peer applications also typically expose APIs to allow the clients to request operations.
- So, the use of an API does not dictate any specific architectural style. However, the SOA architectural style mandates that the services expose APIs so that they can be used.
- microServices also require the exposure of an API to allow them to be used from applications.
- There have been a lot of different approaches to the design of APIs over the years. The RESTful approach is one way of designing an API that can be used in ANY architectural style.

What is a REST*ful* Interface, continued

REST*ful* interfaces are one approach to APIs

- Representational State Transfer is used to define domain data without the need for additional messaging layers or session tracking.
- Representational State Transfer is an approach to API design.
- It does not mandate any specific architectural style.
- Anywhere an API is needed, REST*ful* design approaches can be used.

SOA implementations usually use REST*ful* APIs for their services

- It is NOT required for SOA, but often used in conjunction.
- It is the way services define their APIs.

REST*ful* APIs can be used outside of SOA as well

- A non-SOA application could use RESTful APIs.

What is Virtualization?

Virtualization was an early approach at increasing resource utilization and reducing costs.

Processes running on a real machine do not utilize all of the resources of the hardware very well. When the process is doing I/O operations, the process is usually blocked (sleeping), causing the machine to be idle while the I/O completes. Of course this is simplified, but deploying lots of processes on distributed hardware generally does not utilize that hardware very well, resulting in excess capacity that is underutilized, increasing costs, both in the initial purchase of the resources but also in cooling, electricity, and floor space.

Virtualization was an early approach at increasing resource utilization and reducing costs. Running the processes in a virtual machine, and then hosting multiple virtual machines on a real one, would tend to utilize excess resources better. When one virtual machine was blocked (sleeping), another could be run that could utilize the real hardware. This allow more processes to be run on fewer real machines, reducing costs.

Another benefit to virtualization is that a virtual machine can be moved easily, without affecting the applications running inside it. This provides an easy way to move workload around, such as in a disaster recovery scenario or to recover from an outage or failure.

Virtualization also is applied to the problem of isolating one application service from another. This is another approach that is often used, but not the only one.

What is Virtualization, continued

Virtualization is the creation of simulations of real things

- Virtual machines are simulations of CPUs, memory, and the IO subsystem of a real machine.
- Virtual disks are simulations of a real disk.
- Virtual NICs are simulations of real network interface cards.

Virtualized Simulations

- Virtualized Simulation are isolated from each other. One virtual machine does not share anything with another virtual machine.
- Virtualized Simulations are hosted on a real machine.
- Virtualized Simulations can provide better utilization of real resources (multiple VMs on a real machine).
- Virtualized Simulations compete for real resources much the same as processes running under an Operating System.

What is Virtualization, continued

There are many ways to implement virtualization.

- **Hardware Virtualization:**
  The process of creating virtual in-memory replicas of all hardware. This includes the CPUs, memory, and I/O sub-system (including disks and networks).
  - Virtualizes the hardware and allows any operating system (called a guest) to be hosted.
  - Guest O/S is booted in the virtual machine environment and will behave as usual.
  - This is the approach used by vmWare ESXi servers.
- **Operating System Virtualization:**
  This type of virtualization usually is implemented in the kernel of a host system. The host system then allows multiple operating systems to be created, similar to processes. In effect, each operating system is treated much the same as any other process, competing for resources and is isolated from each other by the host.
  - An operating system is booted on the host, and its kernel contains support for virtualization.
  - The host kernel virtualizes I/O, memory, and other resources which are owned and managed by the host.
  - This is the approach that Lxc, Vagrant, VirtualBox, vmWare VirtualServer, and KVM use.
- **Application or Service Virtualization:**
  This is usually implemented by a container or other management component that virtualizes all of the operating system services such as scheduling, I/O, memory management, and networking. This is used by containers to support whatever environmental dependencies the service may require while allowing it to run on any operating system.
  - Not to be confused with “services” discussed so far, but relative to the services performed by an operating system.
  - Virtualizes the environment, as well as dependencies, that an application needs to execute.
  - This is the approach used by Rkt and Docker.

What is Continuous Integration and Continuous Deployment (CI/CD)?

CI/CD is a philosophy of product development and use that aims to reduce the amount of time to implement new features, decrease costs, improve quality, and enhance the customer experience.

- CI/CD approach to development and deployment of software applications or components is widely applicable, not only to microServices, but to all types of software.
- It came from the adoption of agile methodologies and the need to speed up the delivery of software, automate much of the process, and improve time to market. They are frequently used in agile projects.
- Since microServices are usually developed using agile methodologies, this approach is often used in conjunction.
- It is not essential to using microServices, you could develop and deploy microServices without using CI/CD. However, it is usually considered a best practice, since some of the reasons to use microServices are to improve time to market and reduce development costs, which CI/CD helps with as well.

What is Continuous Integration and Continuous Deployment (CI/CD), continued

Continuous Integration

- As engineers make changes to the components (either by writing new code, changing configuration, or customizing it in some way), the changes are integrated into the code base frequently (or immediately).
- Integration involves:
  - Building the code base
  - Running all tests
  - Running all code metrics (coverage, quality, analysis, etc..)

Continuous Deployment

- The process of taking the package that has been built from the changes and installing them in the appropriate runtime environment
- Deployment involves:
  - Packaging
  - Releasing
  - Publishing
  - Deploying

What is Continuous Deployment Platform (CDP)?

CDP is a collection of tools and processes, templates, and samples that demonstrate a mature development, deployment, monitoring, and support framework for microServices at AT&T.

It uses open-source solutions as well as AT&T developed processes and tools to provide the full lifecycle.

It is not a microService, but aids in the development, testing, packaging, deployment, monitoring, and support of microServices.