Software Deployment-I

Manas Jyoti Das, PhD
Computer Science

Monolith vs Microservices

- Monolithic and microservices are two different approaches to building software applications, each with its own advantages and disadvantages

Monolith vs Microservices

- Monolith
  - Facebook (earlier version)
  - Spotify (earlier version)
  - Many more
- Microservices
  - Amazon
  - Netflix
  - Large complex applications

Monolith

- **Single codebase**: Everything is built as one large system, making it simpler to develop and deploy initially
- **Tightly coupled**: All components are interconnected, so changes in one area can impact other parts of the application (What about different modules?)
- **Scalability challenges**: Scaling the entire application can be difficult and resource-intensive. What do you mean by that?
    - Let's say you created one software of image editing. It is a big software and only one functionality like removing the background is getting used my millions of users. To scale it you have to buy huge server space since you have to scale the whole software not a part of it. Which people are more interested in.

Monolith

- **Limited technology independence**: Changing underlying technologies or languages requires updating the entire codebase
- **“Easier” debugging**: Issues are typically easier to track down due to the centralized nature of the code

Microservices

- **Independent services**: The application is broken down into smaller, independent services that communicate with each other through APIs
- **Loosely coupled**: Changes in one service have minimal impact on others, promoting agility and maintainability
- **Scalability flexibility**: Individual services can be scaled independently based on their specific needs. What do you mean?
    - Building on the example given in the monolith, the background removal service which may not be a large resource intensive service can be spawned many times with a modest increase in the server resources

Microservices

- **Technology diversity**: Different services can use different technologies and languages, promoting flexibility and innovation. Which can also become a curse!!
- **Increased complexity**: Managing and monitoring numerous services can be more challenging
- **Debugging complexity**: Tracing issues across multiple services can be more difficult

Microservices vs Monolith

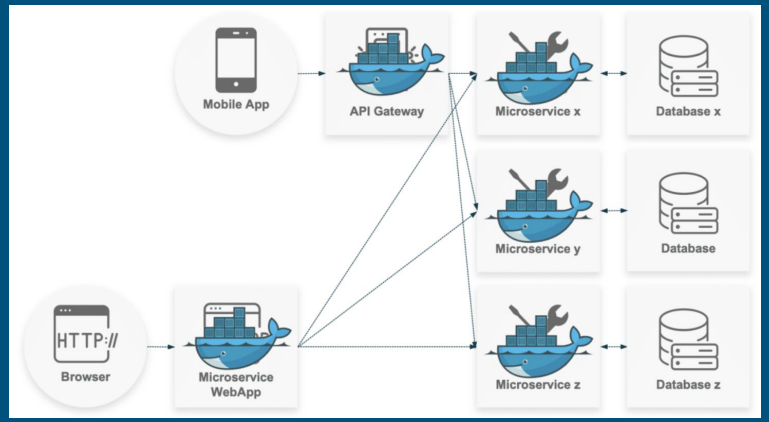
Feature	Monolithic	Microservices
Codebase	Single	Multiple independent services
Coupling	Tightly coupled	Loosely coupled
Scalability	Challenging to scale the entire application	Individual services can be scaled independently
Technology independence	Limited	Flexible, different technologies possible
Debugging	Easier	More complex, requires tracing across services

Microservices vs Monolith

- Choosing the right architecture depends on your specific needs and the complexity of your application. Monolithic architectures might be suitable for smaller, simpler applications, while microservices offer greater scalability and flexibility for larger, more complex systems

Development approach in Microservices

- **1. Containerized Microservices**:
  - Software:
    - Container engine: Docker, containerd, Podman (lightweight virtualized environments)
    - Container orchestration platform: Kubernetes (manages container lifecycle and deployment)
    - Registry: Docker Hub, private registry (stores container images)
  - Benefits: Scalability, flexibility, portability, isolation.
  - Considerations: Increased complexity, learning curve for Kubernetes



Development approach in Microservices

- **2. Serverless Functions**:
  - Software:
    - Serverless platform: AWS Lambda, Azure Functions, Google Cloud Functions (event-driven execution)
    - Monitoring and logging tools: CloudWatch, Azure Monitor, Stackdriver (optional)
  - Benefits: Highly scalable, pay-per-use, no server management.
  - Considerations: Vendor lock-in, limited execution time (Functions typically have limitations on execution duration), cold starts (Initial executions might be slower due to function startup time)

Serverless functions

- When to choose serverless functions
  - Microservices with unpredictable or bursty workloads
  - Event-driven microservices like processing data streams
  - Cost-sensitive applications where idle server costs matter
-

Development approach in Microservices

- **3. Virtual Machines (VMs)**:
  - Software:
    - Cloud provider VM service: AWS EC2, Azure VMs, Google Cloud Compute Engine
    - Configuration management tools: Ansible, Chef, Puppet (automate VM provisioning and configuration)
  - Benefits: Familiar environment, good for complex deployments.
  - Considerations: Less scalable than containers, requires more management.
-

API Gateway

- An API Gateway acts as a single entry point for clients interacting with your microservices architecture
- Routing: Receives client requests and routes them to the appropriate microservices based on predefined rules and logic
- Aggregation: Can combine data from multiple microservices into a single response for the client
- Security: Enforces authentication, authorization, and other security measures
- Traffic management: Implements features like load balancing, rate limiting, and caching to optimize performance and scalability
- Google API Gateway, Kong, Tyk, Apigee, many others

Docker

Download docker

https://docs.docker.com/desktop/install/windows-install/

Before the installation you may have to install wsl (if not already installed)

https://learn.microsoft.com/en-us/windows/wsl/install

Go to the above link and install wsl. It is simple just type `wsl --install` in your command prompt.

Docker

After installation and setup run the command : `docker pull hello-world`

If everything is correctly installed it will pull a docker image from docker hub.

Then run the container that you just downloaded: `docker run hello-world`

Pull another image which contains a slim version of debian which is the basis of ubuntu (linux) operating system: `docker pull debian:bullseye-slim`

To view all the images (containers/pods) that you have downloaded: `docker images`

Running docker container

To run the debian container the command will be: `docker run -it debian:bullseye-slim bash`

- The -it means interactive and bash at the end means you are using a linux version of your command prompt (windows), which is known as bash.

Install packages and do your “stuff”. But remember you are inside linux now windows command will not work. So you are in a “virtual environment” where the OS is linux (in this case debian, basis of ubuntu)

Commiting docker container

After you are done with your programming and “stuff” type exit. You will be back to your windows command prompt (or power window)

The changes that you made inside of the container (like installing software) are not committed. To commit:-

- first: `docker ps -a`
- Second: From the list given as output from the previous command, the latest(top) image is the one that you just exitted. Copy its ID (example: b0b816e15cea)

commiting docker container

Now we will be committing the image whose ID you just copied.

`docker commit ID NAME_YOUR_CONTAINER:VERSION_NUMBER`

Example: `docker commit b0b816e15cea python_doc:v0.1`

Now again do a `docker images` to view your newly created container

To run the newly created container: `docker run -it python_doc:v0.1 bash`

Do not forget to give the version_number, now all your packages and everything that you have done is there.

You can `push` the container to docker hub for people to use. You can make it private too.

Comments

This is not the pro way of doing docker. In practice you have to create a docker script and build the script to create the container (maybe towards the end of the course).

Don’t forget to delete you images that are not required or it will eat up space.

Gateway is open, you may leave now

Thank you