UECS2344 Software Design: Lecture 10

Appendix

Adapted from https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/

Architectural Styles
An *architecture style* is a family of architectures that share certain characteristics. For example, N-tier is a common architecture style. More recently, microservice architectures have started to gain favor.

N-Tier Architecture
N-tier is a traditional architecture for enterprise applications. Dependencies are managed by dividing the application into *layers* that perform logical functions, such as presentation, business logic, and data access. A layer can only call into layers that sit below it. However, this horizontal layering can be a liability. It can be hard to introduce changes in one part of the application without touching the rest of the application. That makes frequent updates a challenge, limiting how quickly new features can be added.

Web-Queue-Worker Architecture
In a **Web-Queue-Worker** architecture, the application has a web front end that handles HTTP requests and a back-end worker that performs CPU-intensive tasks or long-running operations. The front end communicates to the worker through an asynchronous message queue.

Microservices Architecture
If your application has a more complex domain, consider moving to a **Microservices** architecture. A microservices application is composed of many small, independent services. Each service implements a single business capability. Services are loosely coupled, communicating through API contracts.

Each service can be built by a small, focused development team. Individual services can be deployed without a lot of coordination between teams, which encourages frequent updates. A microservice architecture is more complex to build and manage than either N-tier or web-queue-worker. It requires a mature development and DevOps culture. But done right, this style can lead to higher release velocity, faster innovation, and a more resilient architecture.

Event-driven architecture
Event-Driven Architectures use a publish-subscribe model, where producers publish events, and consumers subscribe to them. The producers are independent from the consumers, and consumers are independent from each other.

Consider an event-driven architecture for applications that ingest and process a large volume of data with very low latency, such as IoT solutions. The style is also useful when different subsystems must perform different types of processing on the same event data.

Big Data, Big Compute
Big Data and **Big Compute** are specialized architecture styles for workloads that fit certain specific profiles. Big data divides a very large dataset into chunks, performing paralleling processing across the entire set, for analysis and reporting. Big compute, also called high-performance computing (HPC), makes parallel computations across a large number (thousands) of cores. Domains include simulations, modeling, and 3-D rendering.

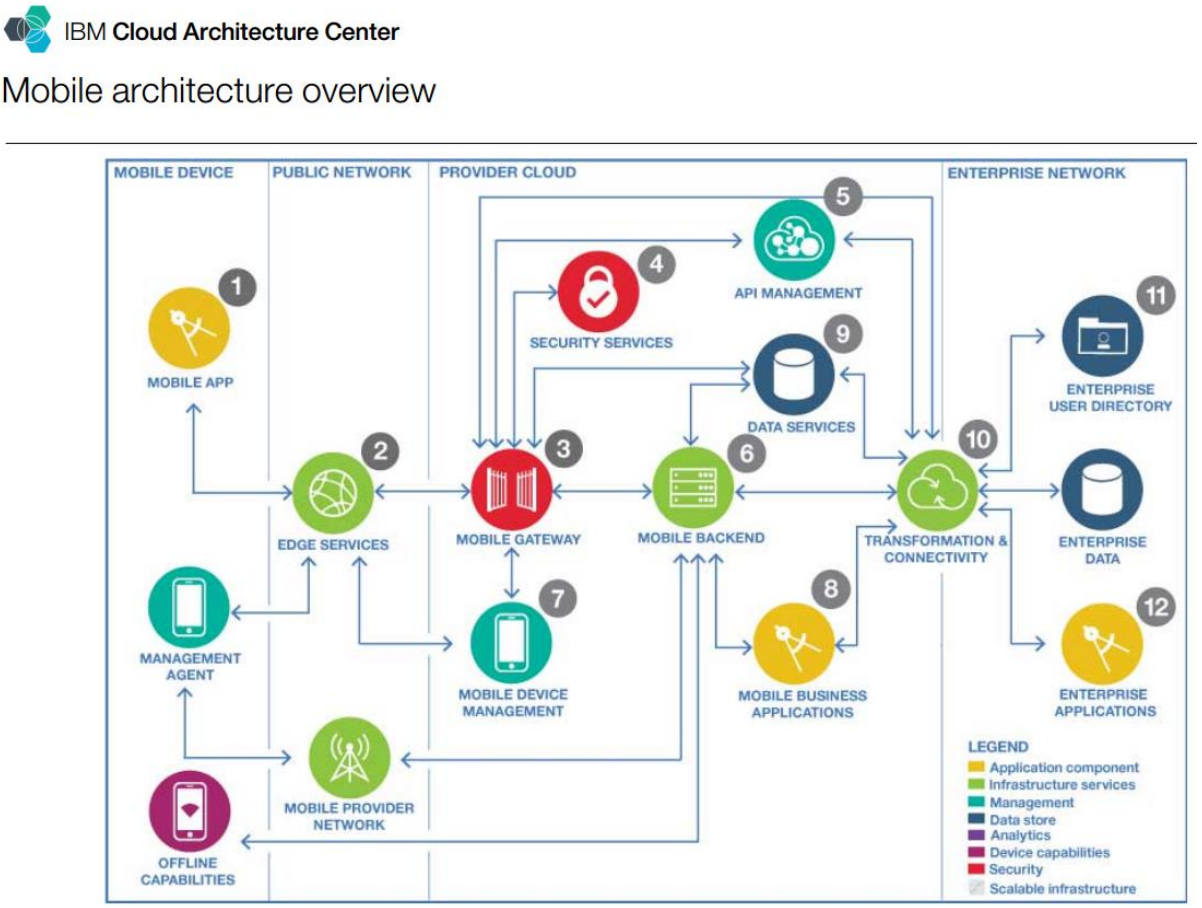
Adapted from https://www.ibm.com/blogs/bluemix/2017/03/introducing-bluemix-developer-console/

What is Cloud Native?

Cloud Native represents philosophies of application development enabled by Cloud technology. These applications take advantage of scalability, flexibility, continuous deployment, DevOps practices, analytics and cognitive capabilities. To incorporate those characteristics into your software lifecycle, it's important to build the applications with a different set of principles than traditional monolith approaches. The IBM Cloud Developer Console makes it easy to create Cloud Native applications that follow modern best practices by generating pre-configured application templates necessary for your microservices architecture.

What makes up a microservices architecture?

Cloud Native applications require building blocks composed of application components, infrastructure and data services. The diagram below is an example of how you could fit those building blocks together to create an application for mobile and web.



With this approach, you can typically spend a lot of time simply setting up each of the microservices and connecting them to other components. The new IBM Cloud Developer Console accelerates this process by scaffolding ready-to-code applications complete with integration to database, cognitive and security services. With our initial launch, we're enabling Mobile Apps, Web Apps, Backend for Frontend, and Microservices. More to come in the future!

What's New?
A developer can now build out Cloud Native applications using our patterns-first approach. You simply select the type of building block you'd like to create followed by which services you'd like to incorporate into your application (i.e., Cloudant database, WatsonConversation, Push Notifications).

After choosing a language, the console will generate a starter project for you to download and edit, deploy locally or to IBM Cloud, and add more services. Not sure which pattern to choose? Here's a short description of those available in our first release.

Mobile App –If you're building a client-side mobile app, start with this pattern. This will enable you to easily integrate with mobile services like Push and Mobile Analytics, Data services like Object Store and Cloudant, and even add security using AppID. The starter will also give you the ability to easily connect with a backend starter.

Web App – If you're building a client-side web app, start with this pattern. Depending on the pattern, you can download runnable code pre-baked with modern web technologies like gulp, sass, react, webpack, and more.

Backend for Frontend – If you're implementing logic to support a mobile or web front end, start with this pattern. This pattern allows you to easily integrate with Data, Security and other Backend patterns.

Microservice –If you're implementing server-side logic for a single function reusable by multiple clients, use this pattern to get your starter code. This will come with a pre-configured manifest.yml or Dockerfile, so you can run it as a CloudFoundry application or in a Docker container anywhere.

Let's talk about building something familiar, a To-Do list application. There are 3 main components to this application: a web front-end responsible for presenting the user with a slick UI, a backend that will hold all the logic, and finally a database to persist all the todo items. Unlike traditional monolith applications, a microservices architecture requires the UI and backend code to be split into two separate applications. This offers many benefits:

1. Allows the UI and the backend to run on different technology stacks – whatever is best for its function.
2. Independent software lifecycle means faster development, deployment, and scaling.
3. Components can be reused and replaced as necessary. If we decide to build a mobile front-end in the future, it can reuse the backend application that already exists.

With The IBM Cloud Developer Console, we don't need to waste hours or days researching tools and frameworks or setting up the stack for each microservice. Instead, we can simply pick a combination frameworks and services from a curated list of starters, then generate and run the code locally or on IBM Cloud using the CLI or web UI.