Micro Service Development Architecture

Micro-services are a software architecture style in which applications are composed of small, independent modules that communicate with each other using well-defined API contracts.

These service modules are highly decoupled building blocks that are small enough to implement a single functionality.

The purpose of microservices architectures is to make it easier to develop and scale applications. Microservice architectures foster collaboration between autonomous teams and enable them to bring new functionalities to market faster.

Two large areas of modern design lend credence to Microservice based design:
Agile/Scrum – agile scrum must be successful via projects being independently built tested and deployed. Otherwise you could not build software iteratively

Cloud hosting – the ability to use flexible services known as **density** combined with the ability to scale their size known as **elasticity** has driven the rise of micro services.

Traditionally software was built as a monolithic architecture. All code was in a single context and project.

The move towards agile consisted of moving to modular design. Various components were built and tested independently. The main application then pulls in these components and use them accordingly.

Micro services takes modular approach one step further. Each service, roles engine, and data layer are built independently. And deployed separately as individual projects. These projects then exchange data via json or xml. It is co-dependent on SOLID design principles in the sense that each module or service is open for extension but closed to modification. We can scale a service as need for that service grows but we do not need to modify it to redo its fundamental use case.

Finally, asynchronous development allows us to consume multiple endpoints simultaneously and increase verbosity and performance through these extensible modules.

AZURE EXAMPLES OF EACH LAYER:
UI Layer – web app, docker containers, virtual machines

Services layer – functions, api app, docker containers, virtual machines, stream analytics, bot services, cognitive services

storage services – documentDB, azure SQL db, storage, data factory.

The whole idea is to develop a single application into smaller services that make up the whole. This development process allows for faster and more verbose application development.

Development patterns for microservices:

Waterfall –

- requirements change (scope creep)
- inflexible
- not tested until end
- Users don't know what they want

order of waterfall:

1. requirements
2. design
3. development
4. test
5. deploy

Preferred development for micro services:
Agile Tenants:

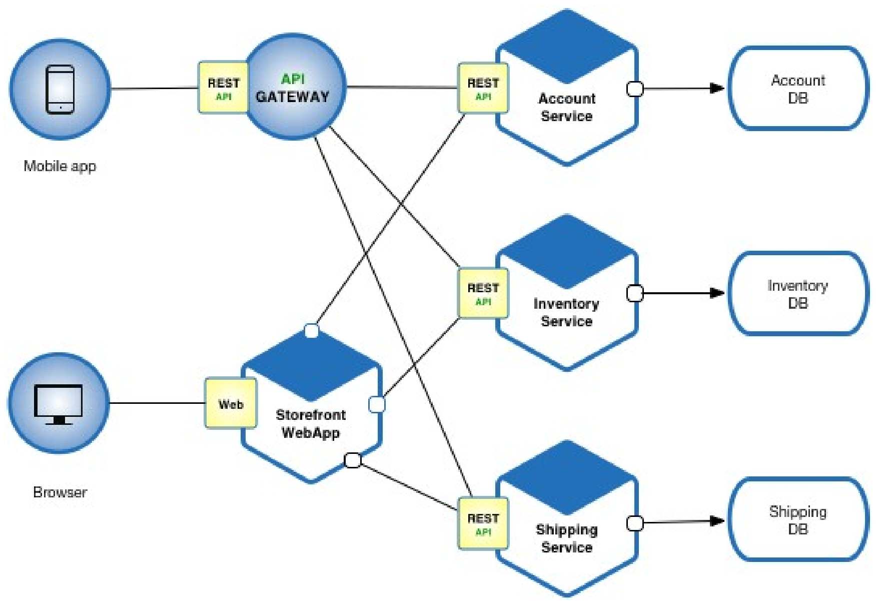
1. individuals and interactions over processes and tools
2. working software over comprehensive documentation
3. customer collaboration over contract negotiation
4. responding to change over following a plan

Rather than performing a big design upfront we break the development cycle down into tinier sprints known as scrum runs. These sprints have a few parts to them:

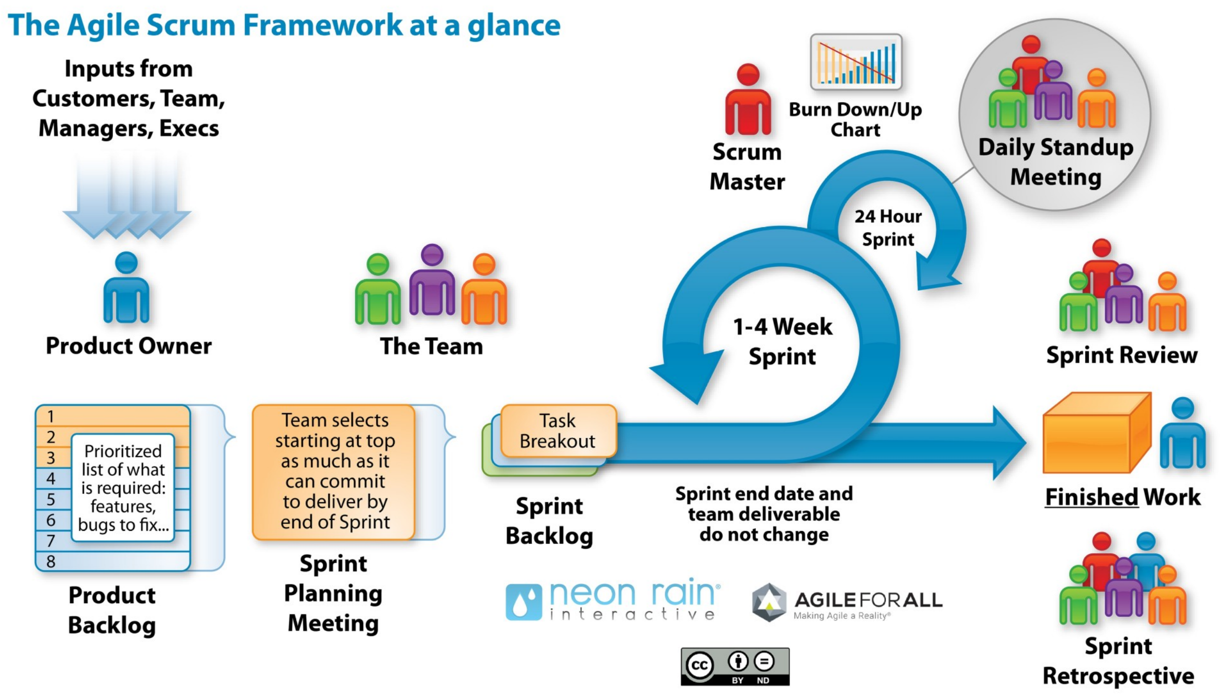
sprint parts:

1. requirements
2. design
3. development
4. test
5. deploy
6. feedback
   (restart at each sprint)

example diagram of microservice based system



example diagram of agile / scrum style development


The Agile Scrum Framework at a glance
Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Burn Down/Up Chart
Daily Standup Meeting
24 Hour Sprint
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Prioritized list of what is required: features, bugs to fix...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective
neon rain interactive
AGILE FOR ALL
Making Agile a Reality
BY ND