Software Design & Architecture
Architectural Styles / Server-Client, Microservices, Serverless

Pengyu Nie

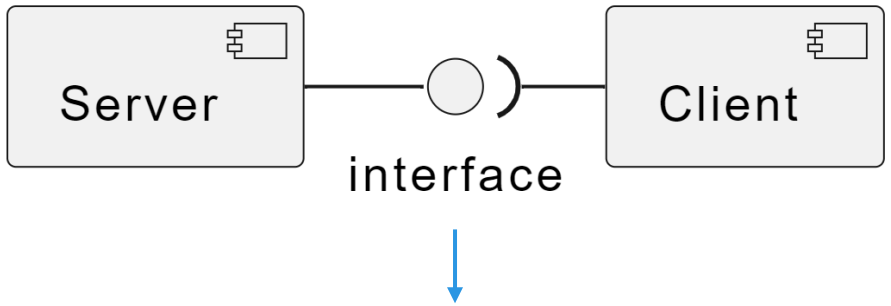
Acknowledgements: slides adapted from previous versions by Mei Nagappan and Shane McIntosh, which are adapted from previous versions by Zhen Ming Jiang, Ahmed E. Hassan, Reid Holmes.

Agenda

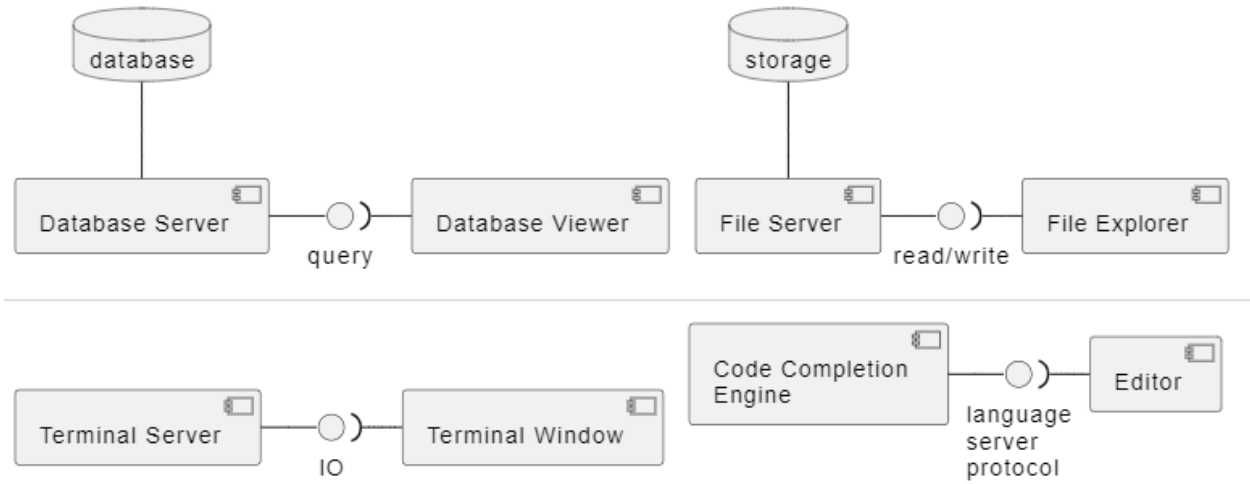
- Server-client
- Microservices
- Serverless

Server-Client

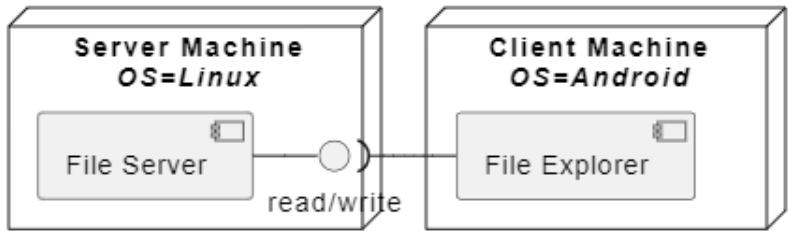
Suitable for applications that involve distributed data and processing across a range of components

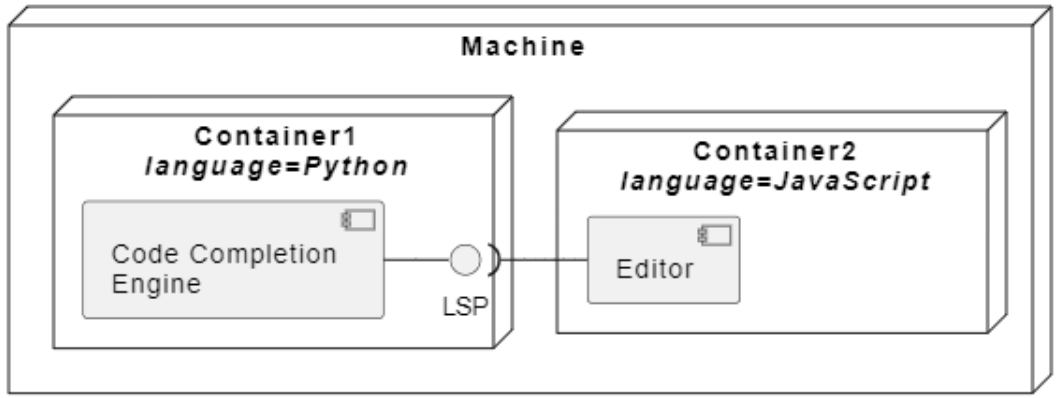


e.g.,
http request
RPC (remote procedure call)
REST API
message queue



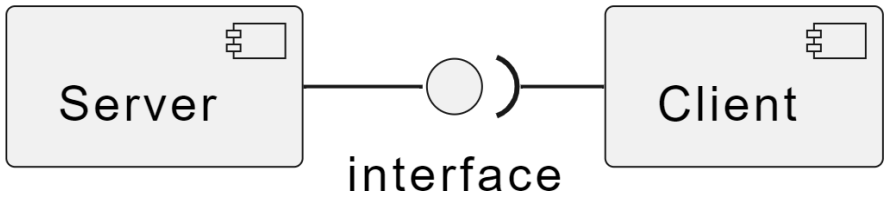
Server-Client – Deployment View

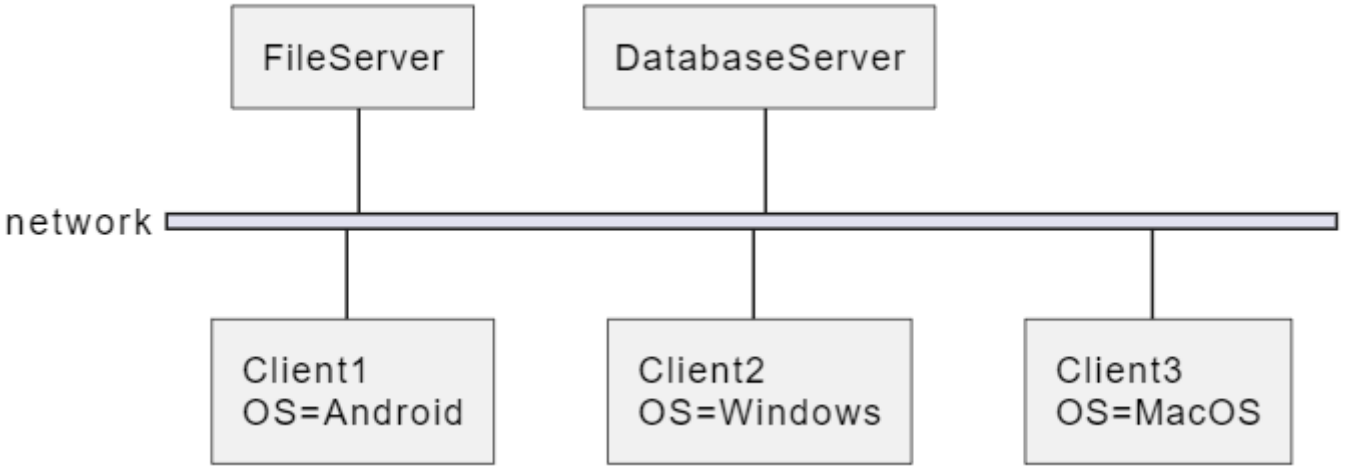




servers and clients can be on the same machine

Server-Client – Network View





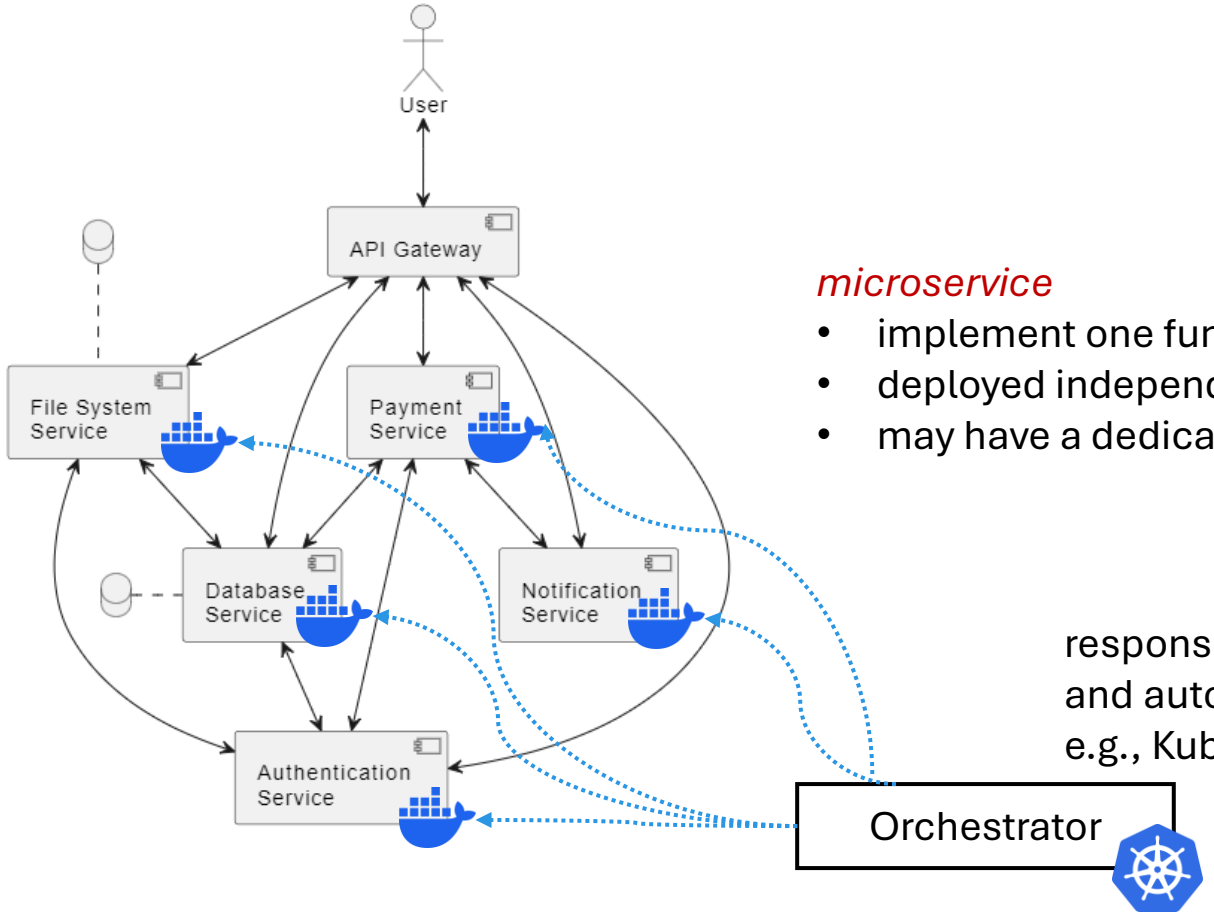
Server-Client – Pros and Cons

+ Straightforward and transparent distribution of data
+ Heterogenerous platforms
+ Easy to add new servers or upgrade existing servers

- No central register of services

Microservices

Suitable for complex applications that require short release cycles and must be highly scalable

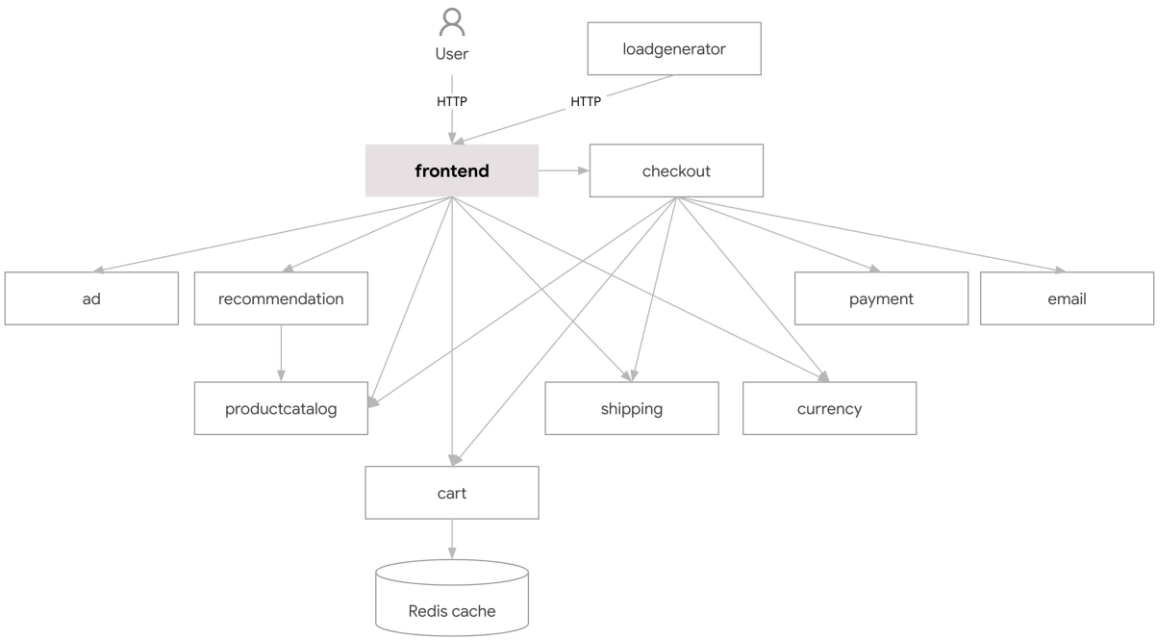


microservice

- implement one functionality (“service”)
- deployed independently, usually in docker containers
- may have a dedicated database

responsible for the deployment, management, and auto-scaling of microservices
e.g., Kubernetes: https://kubernetes.io/

Microservices – Example



Service	Language	Description
frontend	Go	Exposes an HTTP server to serve the website. Does not require signup/login and generates session IDs for all users automatically.
cartservice	C#	Stores the items in the user's shopping cart in Redis and retrieves it.
productcatalogservice	Go	Provides the list of products from a JSON file and ability to search products and get individual products.
currencyservice	Node.js	Converts one money amount to another currency. Uses real values fetched from European Central Bank. It's the highest QPS service.
paymentservice	Node.js	Charges the given credit card info (mock) with the given amount and returns a transaction ID.
shippingservice	Go	Gives shipping cost estimates based on the shopping cart. Ships items to the given address (mock)
emailservice	Python	Sends users an order confirmation email (mock).
checkoutservice	Go	Retrieves user cart, prepares order and orchestrates the payment, shipping and the email notification.
recommendationservice	Python	Recommends other products based on what's given in the cart.
adservice	Java	Provides text ads based on given context words.
loadgenerator	Python/Locust	Continuously sends requests imitating realistic user shopping flows to the frontend.

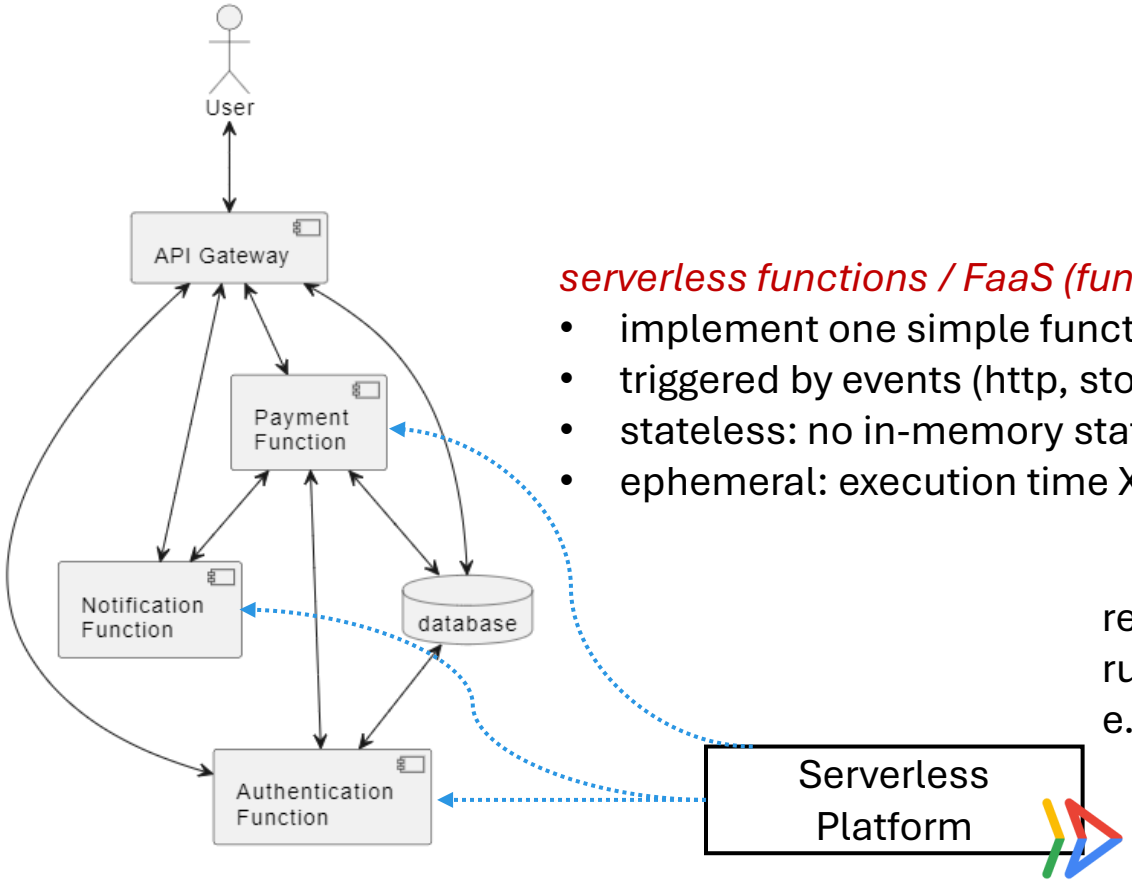
example: https://github.com/GoogleCloudPlatform/microservices-demo

Microservices – Pros and Cons

+ Independently deployable and scalable
+ Reduce downtime
+ Enable a high degree of team autonomy
+ Easier CI/CD integration, simpler maintenance
- High cost (to keep many microservices up and running)
- Complexity

Serverless

Suitable when the system load is not consistent and latency is not a concern



serverless functions / FaaS (function-as-a-service)

- implement one simple functionality
- triggered by events (http, storage events, messages, etc.)
- stateless: no in-memory state between invocations
- ephemeral: execution time X seconds – X minutes

```python
from datetime import datetime

def hello_analytics(data, context):
    """Triggered by a Google Analytics for Firebase log event.
    Args:
          data (dict): The event payload.
          context (google.cloud.functions.Context): Metadata for the event.
    """
    trigger_resource = context.resource
    print(f"Function triggered by the following event: {trigger_resource}")

    event = data["eventDim"][0]
    print(f'Name: {event["name"]}')

    event_timestamp = int(event["timestampMicros"][:-6])
    print(f"Timestamp: {datetime.utcfromtimestamp(event_timestamp)}")

    user_obj = data["userDim"]
    print(f'Device Model: {user_obj["deviceInfo"]["deviceModel"]}')

    geo_info = user_obj["geoInfo"]
    print(f'Location: {geo_info["city"]}, {geo_info["country"]}')
```

responsible for hosting the functions, spawning containers to run them, monitoring events, and auto-scaling
e.g., Google Cloud Run: https://cloud.google.com/functions

more example functions see: https://cloud.google.com/functions/docs/samples

Serverless – Pros and Cons

+ Pay per use
+ Fast deployment and less maintenance
+ Easy to debug

- Third-party dependency
- Initial latency (cold start)
- Stateless nature

Microservices vs. Serverless

Microservices

- Runs 24/7
- In house / on cloud
- Complex functionalities possible
- Expensive upfront

Serverless

- Runs when triggered
- Tied to cloud provider
- Short running simple operations
- Reduced cost

Google Cloud

aws

Azure

Cloud Run

AWS Lambda

Azure Functions

Agenda (recap)

- Server-client
- Microservices
- Serverless

- Setup on Google Cloud:
  - microservices with Kubernetes: https://cloud.google.com/kubernetes-engine?hl=en
  - serverless: https://cloud.google.com/run?hl=en