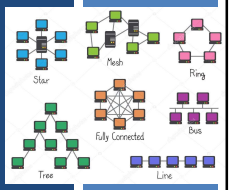
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma



1

OBJECTIVES

- Feedback from 1/16
- Homework 0 – networking review
- Chapter 2.1: Architectural Styles
- Class Activity 2 – Rearchitecting Distributed Systems
- Chapter 2.2: Middleware organization
- Research directions overview

2

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (19 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.05**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.24**

3

FEEDBACK FROM 1/16: WHAT DOES "STATELESS" MEAN?

Stateful vs. Stateless Server Designs

STATEFUL: *server maintains client-specific state*

- Requests from specific clients routed to specific servers holding state
- Keeping state information at server reduces size of messages, allows server to respond more quickly: ***client data already at server***
- Cached client data provides speedup
- Less scalable
- Less fault tolerant (single pt. of failure- clients limited to specific server)

STATELESS: *Server maintains no state information regarding client accesses*

- Requests must contain all required data: ***no memory of client***
- Better fault tolerance: ***server can crash, no state data to loose***
- Where requests are processed DOES NOT MATTER !
- More flexible load balancing
- Better scalability
- Coding stateless server is simpler

4

FEEDBACK - 2

- **Should we apply for "starter account" with $100 credits, or normal AWS account with $75 credits?**
  - Only AWS Educate accounts available via the GitHub Student Developer pack or AWS Educate provide ***ANY*** credits =(
    - ***These accounts no longer require a credit card!***
  - A Normal AWS account only has free tier access, and no credits, but requires a credit card
    - ***These accounts have no service restrictions***

5

ASSIGNMENT 0 - NETWORKING

- Haproxy requires network path to all tomcat docker containers
  - If haproxy can't reach server, routing FAILS
- **Scenario 1 – Use VM Public IPs**
- Configure VM's public IP address and port number in `haproxy.cfg`
- Network traffic is routed out to internet and to public IP
- Goes through AWS firewall (e.g. security group)
- When traffic reaches VM, docker port forwarding rule routes to container
- Ports need to be opened in AWS security group so traffic is allowed to pass from internet to the VM
  - E.g. 8081, 8082, 8083, etc.

6

ASSIGNMENT 0: NETWORKING - 2

- **Scenario 2 - Use VM Private IPs**
- Alternate to using VM's public IP
- Network traffic does not need to go out to the internet. Doesn't leave the VM.
- VM knows its own private IP address (this is observed when typing `ifconfig` command)
- Traffic not routed through network gateway
- VM see this traffic as local traffic.
- Ports probably don't need to be opened in AWS security group
- Requests never leave VM to go through AWS firewall
- Approach is more efficient as network traffic has fewer routing hops

7

ASSIGNMENT 0: NETWORKING - 3

- **Scenario 3 – Use Docker container IPs**
- Use internal Docker container IP addresses
- These IPs are assigned when containers are created
- IPs will vary depending on order of container creation
- Must "shell" into container to check what the IP's are
- Can be done with the following command sequence:

8

```
$ sudo docker run -p 8080:8080 -d --rm tomcat1
56a8e9964feb938fc8e0bd9af0f3a3483e3ca42fb6a428855895c0018bab4d96

$ sudo docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                    NAMES
56a8e9964feb   tomcat1    "/entrypoint_tomcat.…"   2 seconds ago    Up 2 seconds    0.0.0.0:8080->8080/tcp   zealous_kare

# USE THIS COMMAND TO SHELL INSIDE A DOCKER CONTAINER ON THE HOST
$ sudo docker exec -it 56a8e9964feb bash

root@56a8e9964feb:/# apt update
Hit:1 http://archive.ubuntu.com/ubuntu bionic InRelease
. . . Output truncated for brevity

root@56a8e9964feb:/# apt install net-tools
The following NEW packages will be installed:
  net-tools
. . . Output truncated for brevity

# CHECK THE DOCKER CONTAINER'S INTERNAL IP ADDRESS
root@56a8e9964feb:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 95  bytes 202196 (202.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 92  bytes 7111 (7.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

9

ASSIGNMENT 0: NETWORKING - 4

- "ifconfig" command run inside container provides internal IP address
- Command can be installed via the "net-tools" package
- Installation can be added to the Dockerfile
- Drawback to using container IP is that all containers must reside on the same VM (host)

10

ASSIGNMENT 0: NETWORKING - 5

- **NETWORK IP PERFORMANCE TESTING**
- Possible to use "ping" to show how routing via public IP is slower than a private IP or localhost address
- Need to open all ICMP rules in security group
- Pings to the public IP appear about 5x slower
- DO NOT route Amazon VM to VM network traffic using public IPs
  - DATA egress charges apply:
    - First GB outbound transfer is free
    - 9 cent/GB transfer for next 9.999 TB
  - Example: network performance testing with iPerf

11

```
# PING FROM EC2 VM TO EC2 VM PUBLIC IP
$ ping 18.217.177.3
PING 18.217.177.3 (18.217.177.3) 56(84) bytes of data.
64 bytes from 18.217.177.3: icmp_seq=1 ttl=63 time=0.147 ms
64 bytes from 18.217.177.3: icmp_seq=2 ttl=63 time=0.151 ms
64 bytes from 18.217.177.3: icmp_seq=3 ttl=63 time=0.155 ms
64 bytes from 18.217.177.3: icmp_seq=4 ttl=63 time=0.145 ms
--- 18.217.177.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3057ms
rtt min/avg/max/mdev = 0.145/0.149/0.155/0.012 ms

# PING FROM EC2 VM TO LOCALHOST
ubuntu@ip-172-31-14-194:~/docker_tomcat$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.037 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.036 ms
--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.027/0.034/0.039/0.008 ms

# PING FROM EC2 VM TO EC2 VM PRIVATE IP
ubuntu@ip-172-31-14-194:~/docker_tomcat$ ping 172.31.14.194
PING 172.31.14.194 (172.31.14.194) 56(84) bytes of data.
64 bytes from 172.31.14.194: icmp_seq=1 ttl=64 time=0.031 ms
64 bytes from 172.31.14.194: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from 172.31.14.194: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from 172.31.14.194: icmp_seq=4 ttl=64 time=0.034 ms

# PING FROM EC2 VM TO CONTAINER INTERNAL IP ON EC2 VM
$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.042 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.040 ms
```

12

ASSIGNMENT 0: NETWORKING - 6

- **Helpful Tool --> telnet**
- "`telnet`" command provides tool that can test the connectivity to any IP address / port
- Command is already part of Linux, but needs to be installed in a Docker container

```
# apt install telnet
```

- Command to test if container can access IP / port

```
# telnet 172.17.0.2 8080
```


13

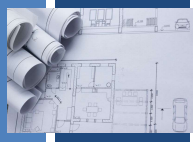
ASSIGNMENT 0: NETWORKING - 7

- When there is a network path, telnet establishs an interactive connection:

```
# telnet 172.17.0.2 8080
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
```

- CAN escape by typing CTRL - right bracket ("] ")
- When no network path exists, telnet simply hangs forever
- Can be killed using key-sequence, CTRL-C (to cancel)

14



CH. 2.1: ARCHITECTURAL STYLES

15

ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

16

DISTRIBUTED SYSTEM GOALS TO CONSIDER

- **Consider how architectural style may impact:**
- Availability
- Accessibility
- Responsiveness
- Scalability
- Openness
- Distribution transparency
- Supporting resource sharing
- Other factors…

17

ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

18

RESOURCE BASED ARCHITECTURES

- Motivation:
  - Increasing number of services available online
  - Each with specific protocol(s), methods of interfacing
  - Connecting services w/ different TCP/IP protocols → integration nightmare
    - Need for specialized client for each service that speaks the application protocol "language"…
- Need standardization of interfaces
  - Make services/components more pluggable
  - Easier to adopt and integrate
  - Common architecture

19

REST SERVICES

- Representational State Transfer (REST)
- Built on HTTP
- Four key characteristics:
  1. Resources identified through single naming scheme
  2. Services offer the same interface
     - Four operations: GET PUT POST DELETE
  3. Messages to/from a service are fully described
  4. After execution server forgets about client
     - Stateless execution

20

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from server
  - Protocol version & status code →
  - Response headers
  - Response body

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

21

REST-FUL OPERATIONS

Operation	Description	
POST	Modify a resource by transferring a new state	(C)reate
GET	Retrieve state of a resource in some format	(R)ead
PUT	Create a new resource	(U)pdate
DELETE	Delete a resource	(D)elete

- Resources often implemented as objects in OO languages
- REST is weak for tracking state
- Generic REST interfaces enable ubiquitous *"so many"* clients

22

EXAMPLE: AMAZON S3

- Amazon S3 offers a REST-based interface
- Requires signing HTTP authorization header or passing authentication parameters in the URL query string
- REST: GET/PUT/POST/DELETE
- SOAP: 16 operations, moving toward deprecation
- Python boto ~50 operations (SDK for Python)
- SDKs for other languages

23

REST - 2

- Defacto web services protocol
- Requests made to a URI – uniform resource identifier
- Supersedes SOAP – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Responses most often in JSON, also HTML, ASCII text, XML, no real limits as long as text-based
- curl – generic command-line REST client: https://curl.haxx.se/

24

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
        namespace="http://www.roguewave.com/soapworx/examples"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
  </service>
</definitions>
```

L5.25

25

```
// REST/JSON
// Request climate data for Washington

{
 "parameter": [
  {
    "name": "latitude",
    "value":47.2529
  },
  {
    "name": "longitude",
    "value":-122.4443
  }
  ]
}
```

L5.26

26

ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

27

PUBLISH-SUBSCRIBE ARCHITECTURES

- Enables separation between processing and coordination
- Types of coordination:

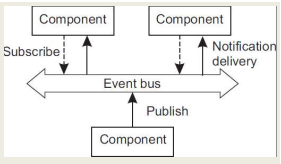
	Temporally coupled (at the same time)	Temporally decoupled (at different times)
Referentially coupled (*dependent on name*)	**Direct** Explicit synchronous service call	**Mailbox** Asynchronous by name (address)
Referentially decoupled (*name not required*)	**Event-based** Event notices published to shared bus, w/o addressing	**Shared data space** Processes write tuples to a shared data space

Publish and subscribe architectures

28

PUBLISH-SUBSCRIBE ARCHITECTURES - 2

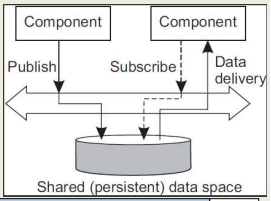
- **Event-based coordination**
- Processes do not know about each other explicitly
- **Processes:**
  - **Publish:** a notification describing an event
  - **Subscribe:** to receive notification of specific kinds of events
- Assumes subscriber is presently up (*temporally coupled*)
- Subscribers must actively **MONITOR** event bus



29

PUBLISH SUBSCRIBE ARCHITECTURES - 3

- **Shared data space**
- Full decoupling (name and time)
- Processes publish "tuples" to shared dataspace (publish)
- Processes provide search pattern to find tuples (subscribe)
- When tuples are added, subscribers are notified of matches
- **Key characteristic:** Processes have no explicit reference to each other

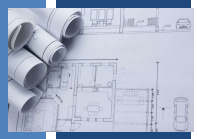


30

PUBLISH SUBSCRIBE ARCHITECTURES - 4

- Subscriber describes events interested in
- Complex descriptions are intensive to evaluate and fulfil
- Middleware will:
- Publish matching notification and data to subscribers
  - Common if middleware lacks storage
- Publish only matching notification
  - Common if middleware provides storage facility
  - Client must explicitly fetch data on their own
- Publish and subscribe systems are generally scalable
- **What would reduce the scalability of a publish-and-subscribe system?**

31



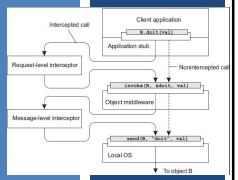
IN-CLASS ACTIVITY: ARCHITECTURAL STYLES

32

DISTRIBUTED SYSTEM GOALS TO CONSIDER

- **Consider how the architectural change may impact:**
- Availability
- Accessibility
- Responsiveness
- Scalability
- Openness
- Distribution transparency
- Supporting resource sharing
- Other factors…

33



CH 2.2: MIDDLEWARE ORGANIZATION

34

MIDDLEWARE ORGANIZATION

- Relies on two important design patterns:
  - Wrappers
  - Interceptors
- Both help achieve the goal of openness

35

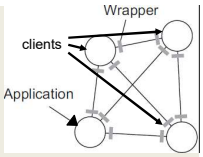
MIDDLEWARE: WRAPPERS

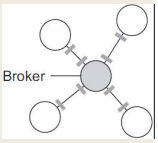
- **Wrappers (also called adapters)**
  - **WHY?:** Interfaces available from legacy software may not be sufficient for all new applications to use
  - **WHAT:** Special "frontend" components that provide interfaces for clients
  - Interface wrappers transform client requests to "implementation" (i.e. legacy software) at the component-level
  - Can then provide modern service interfaces for legacy code/systems
  - Components encapsulate (i.e. abstract) dependencies to meet all preconditions to operate and host legacy code
  - Interfaces parameterize legacy functions, abstract environment configuration (i.e. make into black box)
- Contributes towards system **OPENNESS**
- **Example: Amazon S3:** S3 HTTP REST interface
- GET/PUT/DELETE/POST: requests handed off for fulfillment

36

MIDDLEWARE: WRAPPERS - 2

- Inter-application communication
  - Applications may provide unique interface for every client application
- Scalability suffers
  - N applications → $O(N^2)$ wrappers
- **ALTERNATE: Use a Broker**
  - Provide a common intermediary
  - Broker knows how to communicate with every application
  - Applications only know how to communicate with the broker





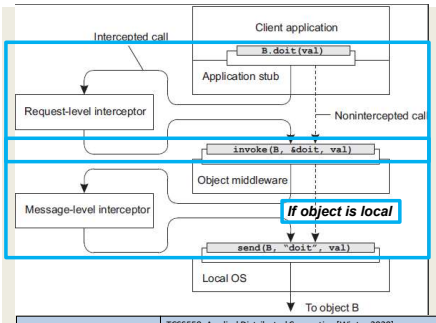
37

MIDDLEWARE: INTERCEPTORS

- **Interceptor**
- Software construct, breaks flow of control, allows other application code to be executed
- Interceptors send calls to other servers, or to ALL servers that replicate an object while abstracting the ***distribution*** and/or ***replication***
  - Used to enable remote procedure calls (RPC), remote method invocation (RMI)
- Object A calls method belonging to object B
  - Interceptors route calls to object B regardless of location

38

MIDDLEWARE: INTERCEPTORS - 2



Request-level interceptor transforms:
`B.doit(val)`
into generic call:
`invoke(B,&doit,val)`

Message-level interceptor in middleware sends message through OS (TCP/IP socket) to transfer data:
`send(B,"doit",val)`

Non-intercepted:

39

MIDDLEWARE INTERCEPTION - METHOD

- **MIDDLEWARE**: Provides local interface matching Object B to Object A
- Object A calls Object B's method provided by local interface
- A's call is transformed into a "*generic object invocation*" by **request-level Interceptor**
- "*Generic object invocation*" is transformed into a **message** by **message-level Interceptor** and sent over Object A's network to Object B
- Interception automatically routes calls to all object replicas

40

MODIFIABLE MIDDLEWARE

- It should be possible to modify middleware without loss of availability
- Software components can be replaced at runtime
- Component-based design
  - Modifiability through composition
  - Systems may have static or dynamic configuration of components
  - Dynamic configuration requires ***late binding***
  - Components can be changed at runtime
- Component based software supports modifiability at runtime by enabling components to be swapped out.
- **Does a microservices architecture (e.g. AWS Lambda) support modifiability at runtime ?**

41

RESEARCH DIRECTIONS

42

CLOUD AND DISTRIBUTED SYSTEMS RESEARCH GROUP

- Meetings on Wednesdays from 12 (12:30) to 1:30pm
- MDS 202
- *MDS is just south of Cherry Parkes*

The CDS group collaborates on research projects spanning Serverless computing (FaaS), Containerization, Infrastructure-as-a-Service (IaaS) cloud, virtualization, infrastructure management, and performance and cost modeling of application deployments. Our research aims to demystify the myriad of options to guide software developers, engineers, scientists, and practitioners to intelligently harness cloud computing to improve performance and scalability of their applications, while reducing hosting costs.

43

QUESTIONS

44

EXTRA SLIDES

45

45