JeroMF
A Software Development Framework for Building Distributed Applications Based on Microservices and JeroMQ

Stephen W. Clyde
Utah State University

Oct. 28, 2019
Motivation

- Distributed applications are pervasive in today’s connected world.
- They can be hard to design, implement, test, deploy, scale, extend, and maintain.
- Service-oriented architectures showed some promise for distributed applications, but applications were still hard to implement, test, deploy, scale, extend, and maintain.
  - Simply splitting an application into multiple independent services can generate more artifacts to manage without necessarily obtaining testability, easier deployment, scalability, etc.
  - In fact, a haphazard refactoring of a distributed application into services may create more complexity.
Problem Statement

Create an open-source framework for distributed applications based on **Microservices** and using **broker-less inter-process communications** that:

- Makes it easier for developers to create secure and reliable distributed applications
- Is flexible, extensible, and scalable
- Can be used for distributed applications running on heterogeneous platforms
- Supports operational tools for managing distributed applications at runtime
Overview

- The Microservice Architecture Style
- Underlying Technology
- Architectural Overview of JeroMQ
- Sample Application
- JeroMF Processes
- JeroMF Services
- JeroMF Communication
- Extension Points
- Evaluation Through Use in Real-world Applications
- Conclusion
The Microservices Architectural Style

Diagram showing the components of the microservices architecture, including API, Business Logic, and Database, connected to various other components.
The Microservices Architectural Style
An interface through which others use the Microservice
The microservice’s logic, which in turn might use other microservices.
The Microservices Architectural Style

An encapsulated persistent store for objects required by the microservice
Characteristics of the Microservices Architectural Style

• Good modularity
  – **Highly cohesive**: each microservice should have a single responsibility
  – **Loosely coupled**: dependencies should be restricted to microservice API’s and documented semantics
  – **Localization of design decisions**: individual design decisions should be encapsulated in one place. The microservices should encapsulate significant design decisions in place one.
  – **Modular reasoning**: A developer should be able to understand a microservice’s functionality by looking at just its implementation.

• Good abstraction and encapsulation

• Testable and Maintainable

• Scalable

• Independently Deployable
Underlying Technology

• Communication Needs
  – Efficient and scalable
  – Reliable and secure
  – Widely available on all common computing platforms
  – Support for synchronous and asynchronous communications
  – Doesn’t require intermediate processes (i.e., No brokers or message servers)

• Security Needs
  – Support for asymmetric and symmetric algorithms
  – Widely available on all common computing platforms

• Choices
  – ZeroMQ as a messaging library
  – BouncyCastle for encryption of application-level communications
ZeroMQ (and JeroMQ)

- ZeroMQ: A message library; not a messaging system
  - High-performance
  - Easy to use
  - Scalable
  - Supports true asynchronous communications
  - Ported to over 40 languages

- Supports multiple transportation layer protocols
  - Transmission Control Protocol (TCP)
  - Inter-process (Pipes)
  - Inter-thread (In-process communications)

- JeroMQ is a native Java port of ZeroMQ

- See [http://zeromq.org](http://zeromq.org) and [https://github.com/zeromq/jeromq-jms](https://github.com/zeromq/jeromq-jms)
Bouncy Castle Crypto API

- Lightweight cryptography API for Java and C#
- A provider for the Java Cryptography Extension (JCE) and the Java Cryptography Architecture (JCA)
- Support for wide range of crypto algorithms.

- The Bouncy Castle Crypto APIs are maintained by an Australian Charity, the Legion of the Bouncy Castle Inc.
- See http://bouncycastle.org
Initial Goals for JeroMF

• Make it easy for developers to
  – Setup containers (processes) of microservices
  – Manage service configurations
  – Create custom services
  – Define and implement reliable application-level communication protocols
  – Authenticate services and encryption application-level communications
  – Track service load and communication statistics
  – Test services and inter-service communications

• It should also allow operators to
  – Gracefully startup and shutdown services
  – Monitor the status of the services in a distributed application
Sample Application

End User's Device
User Interface

Dealer Host Machine
Used-car Server
Get Cars
Get Car Price

Used-car Service
Get Car Implementation
Get Car Price Implementation
Dealer Inventory
JeroMF Processes

- A JeroMF process
  - Is a container for one or more microservices
  - Holds process-level information, e.g. Session values and Settings
  - Does not need to be run on an application server or container platform
  - Can run on any platform with a JVM, including mobile devices
**JeroMF Processes**

- **A JeroMF process**
  - Inherits from `BaseProcess`
  - Is a container for one or more microservices (`BaseService`)
  - Holds process-level information, i.e., Session and Settings
  - Uses a template method pattern to ensure that it is “open for extension but closed to modification”
  - Does not need to be run on an application server or container platform
  - Can run on any platform with a JVM, including mobile devices

```
Settings
+ loadProperties(resourceName : String) : void
+ parse(args : String[]) : void
+ getKeyPairName() : String
+ getPrivatePairPassword() : String
+ getDbUsername() : String
+ getDbPassword() : String

Session
+ getName() : String
+ getZmqContext() : ZmqContext
+ getPrivateKey() : PrivateKey
+ getPublicKey() : PublicKey
+ getPublicKey(keyName : string) : PublicKey
+ getProcessStatus() : RuntimeStatus

BaseProcess
+ addService(service : BaseService) : void
+ getServices() : BaseService[]
+ run() : void
# initialize(args : String[], settingsResourceName : String) : ...
```

![Diagram showing the relationships between Settings, Session, and BaseProcess classes.](image-url)
public class UsedCarServer extends BaseProcess {
public static void main(String[] args){
    UsedCarServer process = new UsedCarServer();
    try {
        process.initialize(args, "server.config");
        UsedCarService service =
            new UsedCarService(instance.getSession(), "UsedCarsService");
        process.addService(service);
        process.run();
    } catch (Exception e) { e.printStackTrace(); }
    finally { process.cleanup(); }
}

@Override
protected Settings createSettings() {
    return new UsedCarSettings();
}
}
JeroMF Services

- **BaseServices** are microservices
  - Without a network interface
  - Can contain business logic
  - Can have connection to a persistent dataset

- **ZmqServices** add to this the ability to
  - Have network-accessible API's
  - Communicate with other services

- All JeroMF Services are active objects

- They use a template method pattern to ensure extensibility through specialization
public class UsedCarService extends ZmqService {

    UsedCarService(Session session, String serviceName) throws ServiceException {
        super(session, serviceName);
    }

    @Override
    protected void initialize() throws ServiceException {
        super.initialize();
        apiResponder.addMessageHandler(ListCars.class, EncryptionMode.None, EncryptionMode.None, msg -> listCars());
        apiResponder.addMessageHandler(GetCarPrice.class, EncryptionMode.None, EncryptionMode.None, msg -> getCarPrice(msg));
    }

    private Message listCars() { ... }
    private Message getCarPrice(Message request) { ... }
}
**HandlerDefinition**

- incomingEncryptionMode : EncryptionMode
- handler : Function<Message, Message>
- outgoingEncryptionMode : EncryptionMode

**Communicator**

- start()
- stop()
- various message send and receive methods
- getStatus() : RuntimeStatus

**Registration Client**

- Requester
  - PassiveRequester
  - Command Responder
  - Active Responder
  - Responder
  - Router
Messages

• JeroMF includes a number of standard messages
  – For registering services
  – For monitoring services
  – For system operations

• JeroMF allows developers to defined their own application specific messages by
  – Specializing Message
  – Defining its serialization
Service Registry

- The Registry is a JeroMF process
- It contains a microservice with the following capabilities
  - Authenticate a service and register it with its public key and API’s
  - Assign a service a symmetric key
  - Look up a service’s public key
  - Look up a service’s API’s
- It can gracefully shut down of a distributed application
- It can periodically require microservice’s to reauthenticate
- Its use is optional
Some Useful Extension Points

- JeroMF processes can contain one or more services, a custom session object, and a custom settings object.
- JeroMF services allow developers to implement all message handlers, business logic, and service synchronization through encapsulated and testable methods.
- JeroMF allows developers to create new kinds of communicators to support custom communication protocols.
- JeroMF allows developers to define and implement virtually any kind of messages.
Evaluation of JeroMF Through Use in Real-world Distributed Applications

- As an initial case study, JeroMF was used to re-design and re-implement the Sync Facility of Utah’s Child Health Advanced Record Management System.
- The Sync Facility monitor changes to various health-care database.
- When changes occur, dependent on the type of change, it may
  - Add or update a child identities in CHARM
  - Cause the re-matching of records across all connected databases
  - Generate or recall alerts
  - Publish changes and alerts to various data consumers
- Its redesign, resulted in 16 different services hosted in 9 processes, not counting the registry
Evaluation of JeroMF Through Use in Real-world Distributed Applications

• The developers reported that
  – Designing the system bases on single-responsibility microservice was easy in some areas, but challenging in others.
    – Achieving tight cohesion doesn’t always come naturally.
  – Each service was relative easy to implement; JeroMF took care of the distribution details.
  – Testing each service was relative easy, since only the setup and business logic needed to be tested.
  – The reliability provided by the new Sync Facility is better than in the old version.
  – The security in the new Sync Facility is better than in the old version
    – The developers added service-level authentication and encrypted communications at the application level, with virtual no extra effort.
  – Deploying the new Sync facility is easier than the old version.
Future Work

• Other BaseService specializations to support other different messaging libraries:
  – HttpService
  – JmsService

• Extensible services that will act as request proxies and load balancers.

• Improved deployment and scalability features

• Empirical studies and qualitative analyses that will more rigorously evaluate its utility, reusability, extensibility, scalability, security, reliability, and maintainability.

• The tracking of software problem reports, time to resolution, induced errors from bug fixes, etc.
Conclusion

• So far, our experience with JeroMF has been positive
  – BaseProcess makes it easy to define new service containers.
  – ZmqService makes it easy to create custom microservices that can implement diverse and sophisticated functionality.
  – The predefined Communicator and Message classes allow developers to implement common styles of communication.
  – They also provide excellent starting points for implementing application-specific communication protocols.
  – It easy for developers to use either asymmetric or symmetric encryption.
  – The optional Registry process can act like a key store for the public keys of registered services, simplifying key management.

• We would welcome collaborators who would like to help refine and expand JeroMF
Questions?