WCC|2019 | THE COSMOPOLITAN™ of LAS VEGAS

Craig Drabik
presents

Aviator Distributed Ledger Platform
Developing Hashgraph Applications with Aviator Core Framework

About Me

Craig Drabik

- Technical Lead, TxMQ Disruptive Technologies Group
- Architect, Developer, Designer
- Web, Mobile, Blockchain/DLT, Enterprise Software

TxMQ, Inc.

- Founded in 1979
- Application Development
- Middleware and Integration
- Staffing

Swirlds Hashgraph

- Hashgraph is a a DAG-based aBFT fault tolerant consensus mechanism
- As transactions are distributed throughout the network via gossip, the network builds a graph that contains the history of when each node learned of each transaction. This data is used to calculate how nodes will vote on the order of transactions without actually collecting votes.
- It is leaderless, asynchronous, and provides fair ordering of transactions across the network with low overhead and 100% finality in seconds.
- Swirlds provides a Java SDK for developers to build private ledger applications using Hashgraph consensus.

Aviator Platform

The Aviator Platform is a suite of tools for developing, scaling, and managing enterprise distributed ledgers.

- Aviator Navigator: Node codebase including smart contracts, data integration components, and swappable consensus
- Aviator Runway: Development and testing tools for designing, building, and testing distributed ledger networks
- Aviator Control Tower: Configure, deploy, monitor, and manage Aviator networks

Aviator Core Framework

TxMQ's Aviator Core Framework makes it easier for developers to build DLT applications using the Swirlds SDK.

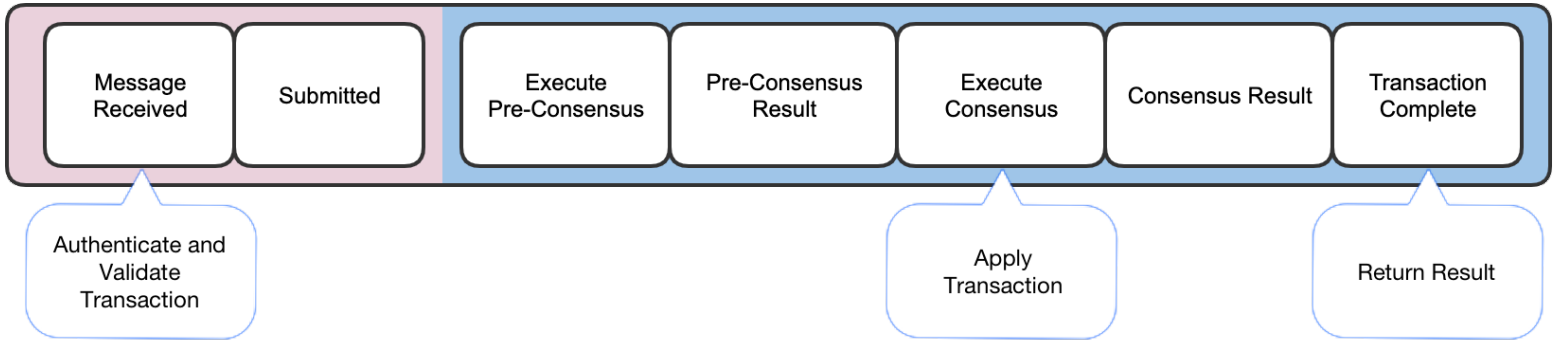
- Event-driven application programming model
- REST and Web Socket integration
- Blockchain-based transaction logging and history
- Designed to be familiar to developers with an enterprise Java background. If you know what an annotation is and can run a Maven build, you can develop applications with Aviator Core Framework.

Pipeline Programming Model

Aviator applications are developed using an event-driven programming model called the pipeline. Every transaction follows the same pre-determined event flow.

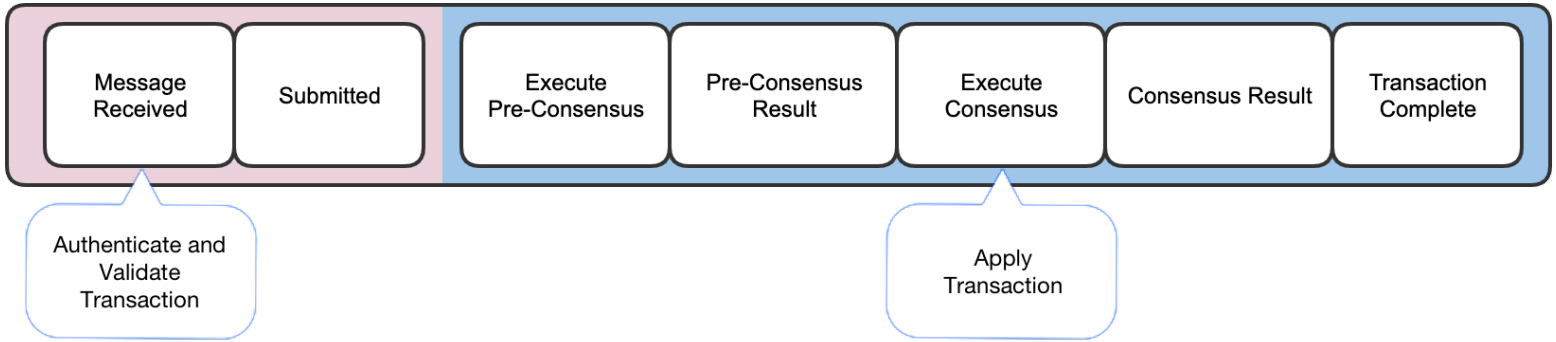
Pipeline Programming Model

Events are issued as transactions move from one stage to the next. Applications listen for these events in order to take action or report status for a given transaction.



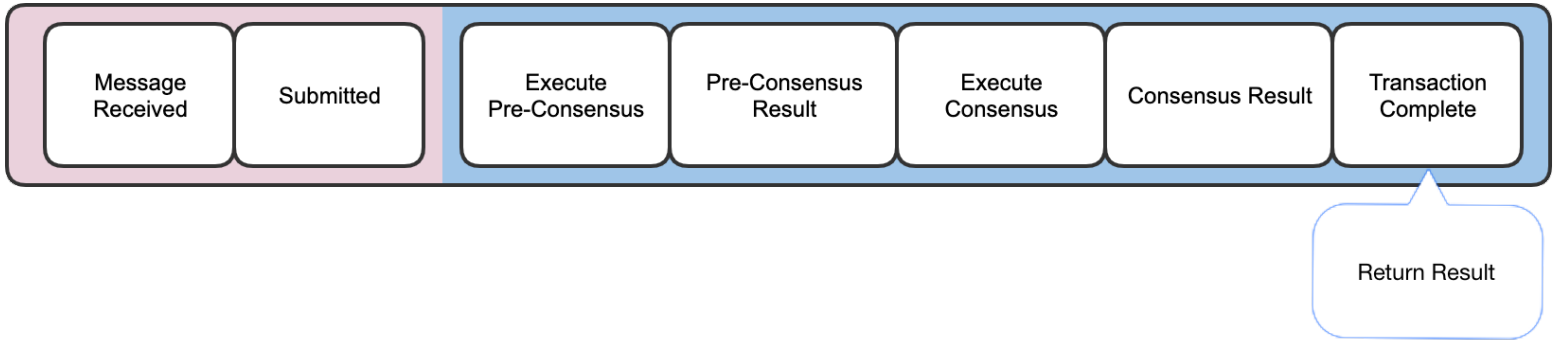
Pipeline Programming Model

There are two types of event listeners in an Aviator application: Handlers and Subscribers. Handlers”do work” inside the application, such as validating or applying transactions.



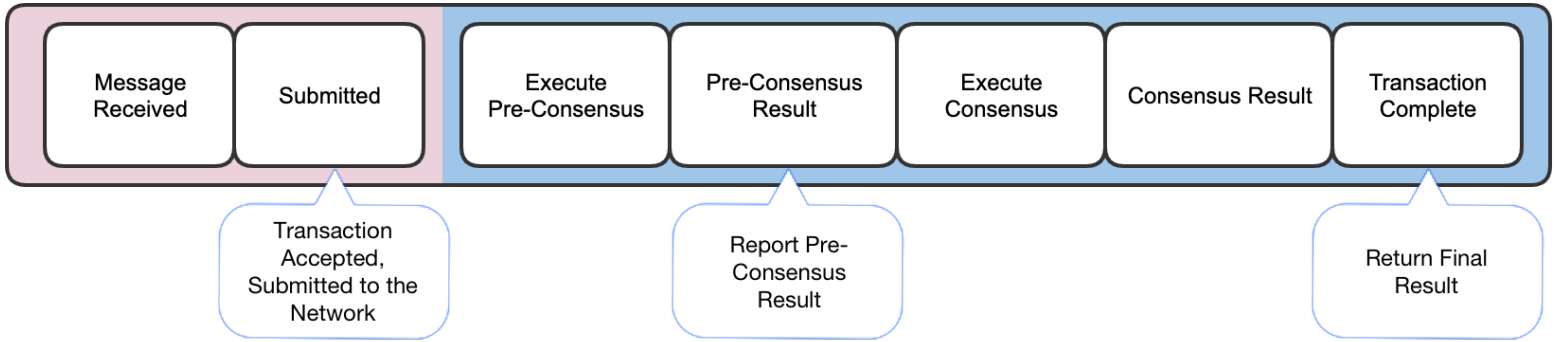
Pipeline Programming Model

Subscribers report status or results back to a calling application. Applications integrated through REST can return only one status…



Pipeline Programming Model

…while applications integrated through web sockets can return multiple statuses at different stages in the pipeline.



Transaction Types

Each type of transaction in an ACF application is represented by a transaction type key. Keys are defined in code using annotations. At startup, the framework builds a list of valid transaction types from annotated classes.

```
@TransactionTypes(namespace=ZooDemoTransactionTypes.NAMESPACE, onlyAnnotatedValues=true)
public class ZooDemoTransactionTypes {
        public static final String NAMESPACE = "ZooDemoTransactionTypes";

        @TransactionType
        public static final String GET_ZOO = "GET_ZOO";

        @TransactionType
        public static final String ADD_ANIMAL = "ADD_ANIMAL";
}
```

Each type of transaction in an ACF application is represented by a transaction type key. Keys are defined in code using annotations. At startup, the framework builds a list of valid transaction types from annotated classes.

Accepting Requests

Aviator Core Framework includes a bundled web server that can host JAX-RS REST services. If you're using web sockets to communicate, you can skip this step.

```
@POST
@Path("/zoo/animals")
@Produces(MediaType.APPLICATION_JSON)
public void addAnimal(Animal animal, @Suspended final AsyncResponse response) {
    AviatorMessage<Animal> message = new AviatorMessage<Animal>(
            new AviatorTransactionType(
                ZooDemoTransactionTypes.NAMESPACE,
                ZooDemoTransactionTypes.ADD_ANIMAL),
            animal
    );
    this.subscriberManager.registerResponder(message, ReportingEvents.transactionComplete, response);
    try {
        message.submit();
    } catch (Exception e) {
        response.resume(Response.serverError().entity(e).build());
    }
}
```

Applying Business Logic

Handlers apply business logic at specific points in the pipeline. Handlers are decorated with @AviatorHandler, indicating the transaction type and pipeline stages they are invoked for.

```
@AviatorHandler(namespace=ZooDemoTransactionTypes.NAMESPACE,
                transactionType=ZooDemoTransactionTypes.ADD_ANIMAL,
                events={PlatformEvents.executePreConsensus, PlatformEvents.executeConsensus},
                payloadClass=Animal.class)
public void addAnimal(AviatorMessage<Animal> message, SocketDemoState state) {
    //todo:  improve this so that we're testing if an animal of the same name exists, and failing if so
    Animal animal = message.payload;
    switch (animal.getSpecies()) {
        case "lion":
            state.addLion(animal.getName());
            break;
        case "tiger":
            state.addTiger(animal.getName());
            break;
        case "bear":
            state.addBear(animal.getName());
            break;
    }
}
```

Reporting Results

Subscribers report information about the status of a transaction. Subscribers are decorated with the @AviatorSubscriber annotation, which describes the transaction type and pipeline stages they are invoked for.

```
@AviatorSubscriber(namespace=ZooDemoTransactionTypes.NAMESPACE,
                   transactionType=ZooDemoTransactionTypes.ADD_ANIMAL,
                   events= {
                           ReportingEvents.submitted,
                           ReportingEvents.preConsensusResult,
                           ReportingEvents.consensusResult,
                           ReportingEvents.transactionComplete })
public void addAnimalTransactionProgress(AviatorNotification<?> notification) {
    String myName = PlatformLocator.getState().getMyName();
    System.out.println("Sending notification from " + myName);
    this.sendNotification(notification);
}
```

Querying Data

Handlers and Subscribers pair up to query data and return results. The return value of a handler is propagated to the next pipeline stage so that subscribers can report it back to client applications.

Querying Data

```
@AviatorHandler(namespace=ZooDemoTransactionTypes.NAMESPACE,
                transactionType=ZooDemoTransactionTypes.GET_ZOO,
                events={PlatformEvents.messageReceived})
public Zoo getZoo(AviatorMessage<?> message, SocketDemoState state) {
    Zoo zoo = new Zoo();
    zoo.setLions(state.getLions());
    zoo.setTigers(state.getTigers());
    zoo.setBears(state.getBears());

    message.interrupt();
    return zoo;
}
```

```
@AviatorSubscriber(     namespace=ZooDemoTransactionTypes.NAMESPACE,
                   transactionType=ZooDemoTransactionTypes.GET_ZOO,
                   events={ReportingEvents.transactionComplete})
public void getZooTransactionCompleted(AviatorNotification<?> notification) {
    AsyncResponse responder = this.getResponder(notification);
    if (responder != null) {
        responder.resume(notification);
    }
}
```

What's Next: ACF and Hedera Consensus Service

- Hedera is a public ledger based on Swirlds Hashgraph consensus
- Hedera Consensus Service provides a mechanism that uses the Hedera public network to order transactions for off-ledger applications.
- TxMQ plans to support Hedera Consensus Service in the Aviator Core Framework
- Applications developed on Aviator Core Framework today can use Hedera Consensus Service with no code changes

More Information

- Aviator Core Framework: https://github.com/TxMQ/aviator-core
- ACF Demo Application: https://github.com/TxMQ/aviator-zoo-demo
- Aviator Platform: https://dtg.txmq.com
- Swirlds Hashgraph: https://swirlds.com
- Hedera Public Network: https://hedera.com
- TxMQ DTG: https://dtg.txmq.com