Automatically generating microservices architectures from user stories

PLANT: User Story Toolchain

Quinten Coltof
Ana Oprescu
Thomas van Binsbergen

April 20, 2023

Research questions

How can we design a system that generates an MSA from a (structured text) user story?

RQ1 How can a user define data computations and machine learning models in a natural and declarative way (user story)?

RQ2 How, given a user story, can we generate a microservices architecture using basic building blocks?

RQ3 Which characteristics are important for the generated microservices architectures?

RQ4 How can we optimize the generated microservices architecture based on these characteristics?

PLANT: Goal

Express the users (non-technical) goal, automatically resolving all constraints.

PLANT: Goal

Express the users (non-technical) goal, automatically resolving all constraints.

Constraints

- Automatic integration
- Adhere to resource constraints
- Ensure quality of service
- Adhere to legal contracts

Framework

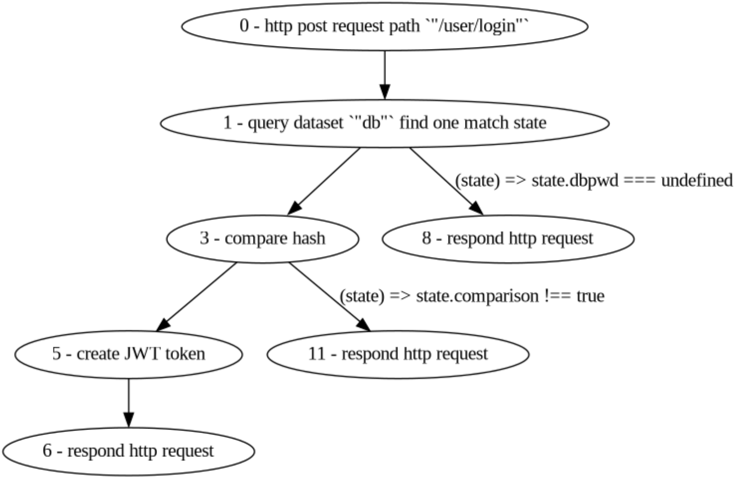
1. From natural language to YAML
2. From YAML to configuration
3. Deployment
4. Runtime environment

Login system ideal

As a user, I want to login successfully when I supply the correct username and password.

Login system current

Login pipeline as a suite of user stories.


0 - http post request path `"/user/login"`
1 - query dataset `"db"` find one match state
(state) => state.dbpwd === undefined
3 - compare hash
8 - respond http request
(state) => state.comparison !== true
5 - create JWT token
11 - respond http request
6 - respond http request

Login system 1

```
given: a http post request with path "/user/login"
       on port 3000 with parameter "username" of
       type "string" and a parameter "password"
       of type "string"
then: # { username: "john", password: "pwd" }
  - pre:
     select:
     - username
    # { username: "john" }
    do: query dataset "db" find one match state
    post:
      upsert:
      - password as dbpwd
      - _id as uid
    #{ username: "john", password: "pwd",
    #  dbpwd: "£2a£12£abc", uid: 1 }
```

Login system 2

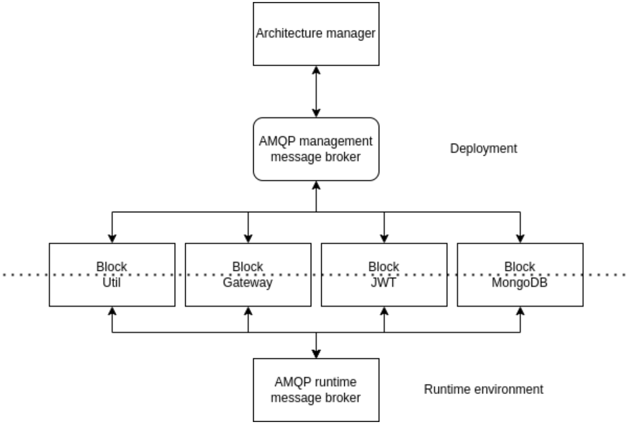
```
# { username, password, dbpwd, uid }
  - pre:
      select:
      - password
      - dbpwd
    # { password: "pwd", hash: "£2a£12£abc" }
    do: compare hash
    post:
      set: comparison
      unset:
      - password
      - dbpwd
# { username: "john", uid: 1, comparison: true }
```

Login system 3

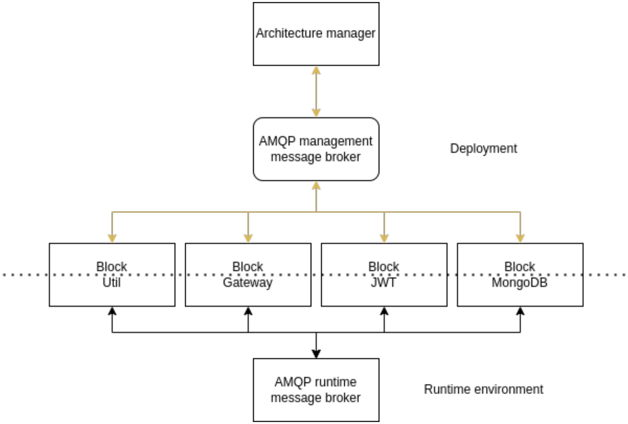
```
# { username, uid, comparison }
- given: comparison not equal `true`
  then:
  - pre:
      select:
        - '"Incorrect password" as body'
        - '`401` as status'
    #{ body: "Incorrect password", status: 401 }
    do: respond to the http request on port 3000
  - stop
```

Deployment and runtime environment

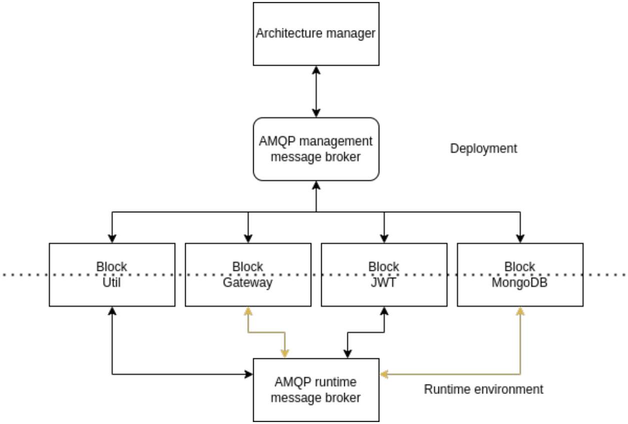
Microservices architecture



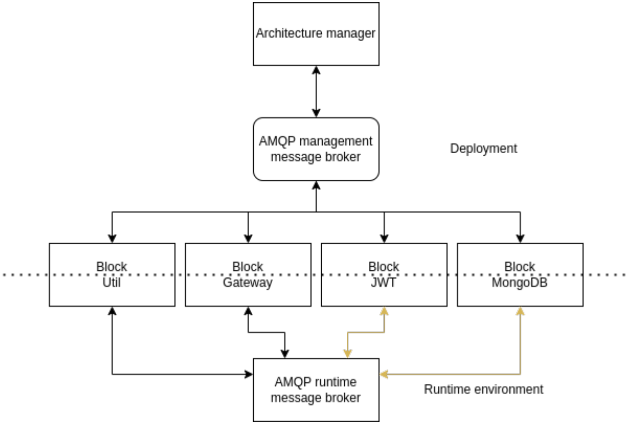
Microservices architecture: Deployment



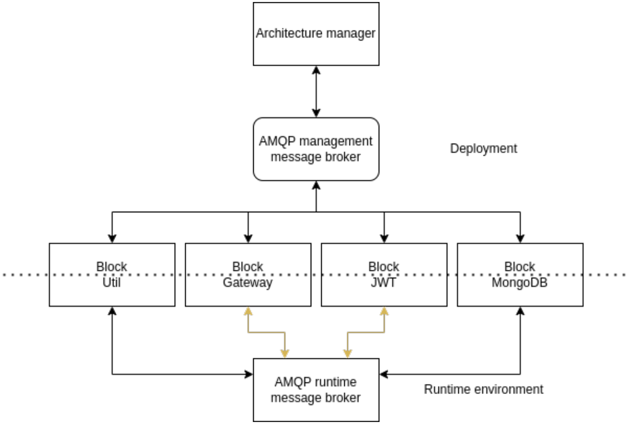
Microservices architecture: Example



Microservices architecture: Example



Microservices architecture: Example



Conclusion

Research questions

RQ1 How can a user define data computations and machine learning models in a natural and declarative way (user story)?

RQ2 How, given a user story, can we generate a microservices architecture using basic building blocks?

RQ3 Which characteristics are important for the generated microservices architectures?

RQ4 How can we optimize the generated microservices architecture based on these characteristics?

Framework

- From natural language to YAML
- From YAML to configuration
- Deployment
- Runtime environment

Appendix ahead

From natural language to configuration

Input

a http post request with path ”/user/login” on port 3000 with parameter ”username” of type ”string” and a parameter ”password” which is of type ”string”

Tokenized and Lemmatized

http post request path ‘”/user/login”‘ port 3000 parameter ‘”username”‘ of type ‘”string”‘ and parameter ‘”password”‘ of type ‘”string”‘

Parsed

http post request path ‘”/user/login”‘ port 3000

1. parameter ‘”username”‘ of type ‘”string”‘
2. and parameter ‘”password”‘ of type ‘”string”‘

x+2-10

```
name: x+2-10
endpoint: amqp://rabbitmq
datasets:
userStories:
- given: a http get request path "/" port 3000
         parameter "input" of type "number"
  then: # { input: 10 }
  - pre:
      select:
      - input as a
      - '`2` as b'
    do: plus # { a: 10, b: 2 }
    post:
      set: res
  - pre: # { input: 10, res: 12 }
      select:
      - res as a
      - '`10` as b'
    do: minus # { a: 12, b: 10 }
  - respond to the http request on port 3000 # 2
```

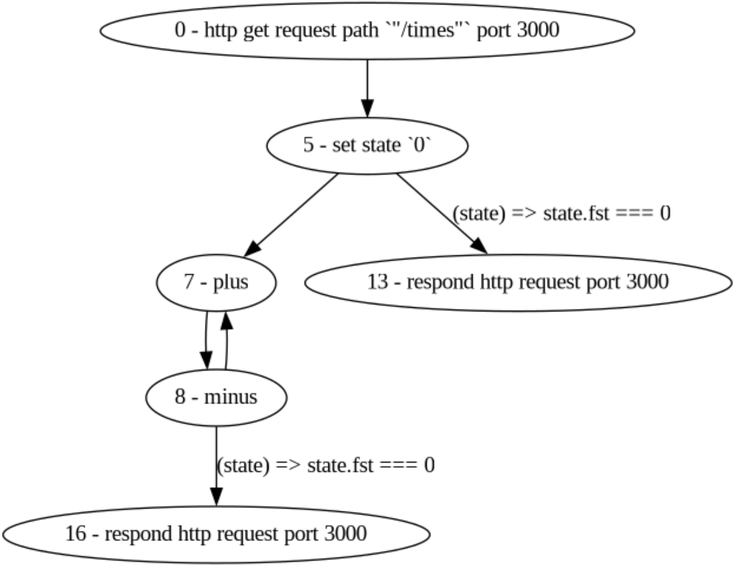
For loop

```
name: For loop
endpoint: amqp://rabbitmq
datasets:
userStories:
- given: |
    http get request path "/times" port 3000 parameter "fst" of
    type "number" and parameter "snd" of type "number" and parameter
    "operation" of type "string"
  then:
    - do: set state `0`
      post:
        set: res
    - pre:
        select:
        - res as a
        - snd as b
      do: plus
      post:
        set: res
    - pre:
        select:
        - fst as a
        - '`1` as b'
      do: minus
      post:
        set: fst
    - given: fst equals `0`
      then:
      - pre:
          pick: res
        do: respond to the http request on port 3000
      - stop
    - goto 3
```

```
def multiply(fst, snd):
    res = 0

    3: res = res + snd
    fst = fst - 1
    if fst == 0:
      respond res
      stop
    goto: 3
```

For loop Control Flow Graph


0 - http get request path `"/times"` port 3000
5 - set state `0`
(state) => state.fst === 0
7 - plus
13 - respond http request port 3000
8 - minus
(state) => state.fst === 0
16 - respond http request port 3000

IO Library

```
import MSAMessaging from '@amicopo/msamessaging';

const io = new MSAMessaging();

io.register('min', ({ input: { a, b } }) => a - b)
io.register('plus',({ input: { a, b } }) => a + b)

io.register('log', ({ input }) => {
  console.log(input);
  return input;
})

io.start();
```

```
{ "archEndpoint": "amqp://rabbitmq", "archExchange": "arch-management-util" }
```