CloudBunch

Концепция сотворения масштабируемых
Cloud Ready
сверх-приложений.

Или, как натянуть сову на глобус,
отделяя мух от котлет.

Потребность

- Широкий спектр бизнес-процессов требует автоматизации (работы много)
- В общем случае бизнес-процесс - это многошаговый и разветвлённый диалог (работа сложная)
- Объективная реальность подвержена изменениям, необходимо обеспечить короткий Time-to-market (делать надо быстро)
- Необходимо быть готовыми к увеличению нагрузки при разумных затратах на аппаратное обеспечение (делать надо надёжно но в рамках бюджета)
- Click to add text

Исходя из потребности

- Необходимо иметь возможность декомпозировать большие приложения на множество независимых компонентов для
  - Разделения ответственности
  - Упрощения внутреннего устройства компонентов
  - Обеспечения независимого цикла разработки, тестирования и сопровождения
  - Унификации и переиспользования частей автоматизируемых процессов
- Для этого крайне желательно, чтобы
  - Взаимодействие между компонентами происходило на основе строгих контрактов
  - Компоненты были горизонтально масштабируемы (Cloud ready)
  - Система самодокументировалась (Список компонентов, контракты, форматы, взаимосвязи в реальном времени)

Stateless микросервисы способны обеспечить всё кроме последнего

Постановка

- Для автоматизации многошаговых, разветвленных бизнес-процессов, и обеспечения при этом короткого Time-to-market

  (*чтобы делать много сложной работы быстро и результат был надёжным*)

  необходимо иметь возможность строить иерархические **State Machine** из независимых **Stateless** микросервисов.

Существующие решения

- Click to add text

Disclaimer

- Излагаемая далее концепция и библиотека, созданная на её основе, не накладывают ограничений на
  - Транспортные протоколы взаимодействия между клиентом и роутером, а так-же между роутером и Flow компонентом
  - Контракты взаимодействия между компонентами – на усмотрение разработчика компонента
  - Контракты взаимодействия с клиентом – на усмотрение разработчика компонента
  - Состав и структуры данных о состоянии компонентов – так-же на усмотрение разработчика
- Дальнейшее содержимое излагает точку зрения автора, автор не несёт ответственности за целостность и полноту изложения а также за целостность и неизменность Вашей картины мира
- **Дальнейшее содержимое может вызвать дискомфорт у некоторых категорий зрителей**

Концепция

ведущий замысел, конструктивный принцип

- Для построения State Machine необходимо где-то сохранять состояние
  - Локальная память – не применимо для Stateless микросервисов
  - Распределенный кэш – приемлемо
- Для построения иерархической State Machine нужно где-то хранить стек и обеспечивать взаимодействие между компонентами
  - **Распределенный кэш** – не лучшее место для сохранения разделяемой информации, - неявная передача данных, сильная связанность – **распределённый монолит**

выделение функции управления

- Можно **вынести** хранение состояний, стека и **управление** связями **в** отдельный **управляющий микросервис**, который может пользоваться распределённым кэшем
- Исполнительные компоненты при этом становятся чистыми микросервисами и получают всё что нужно, влючая состояние, в аргументе
- **Декомпозировать** такую систему удобно **до** уровня **State Machine**

  (Flow компонента)

Стратегия
(стр. 1)

- Система разбивается на Flow компоненты - микросервисы, каждый из которых реализует плоскую **State Machine,** состоит из набора **State** и описывает для себя переходы между ними
- Flow компонент отвечает строго на один вопрос и явно декларирует контракт для обращения из других компонентов системы
- Каждый State задаёт строго один вопрос либо клиенту, либо другому Flow компоненту и явно декларирует контракт взаимодействия с клиентом или контракт вызова другого компонента системы
- State не зависит от Flow, в котором он работает
- Flow зависит от набора State, из которых он составлен
- **Все необходимое для работы (включая свое состояние) компонент - микросервис получает в аргументе**

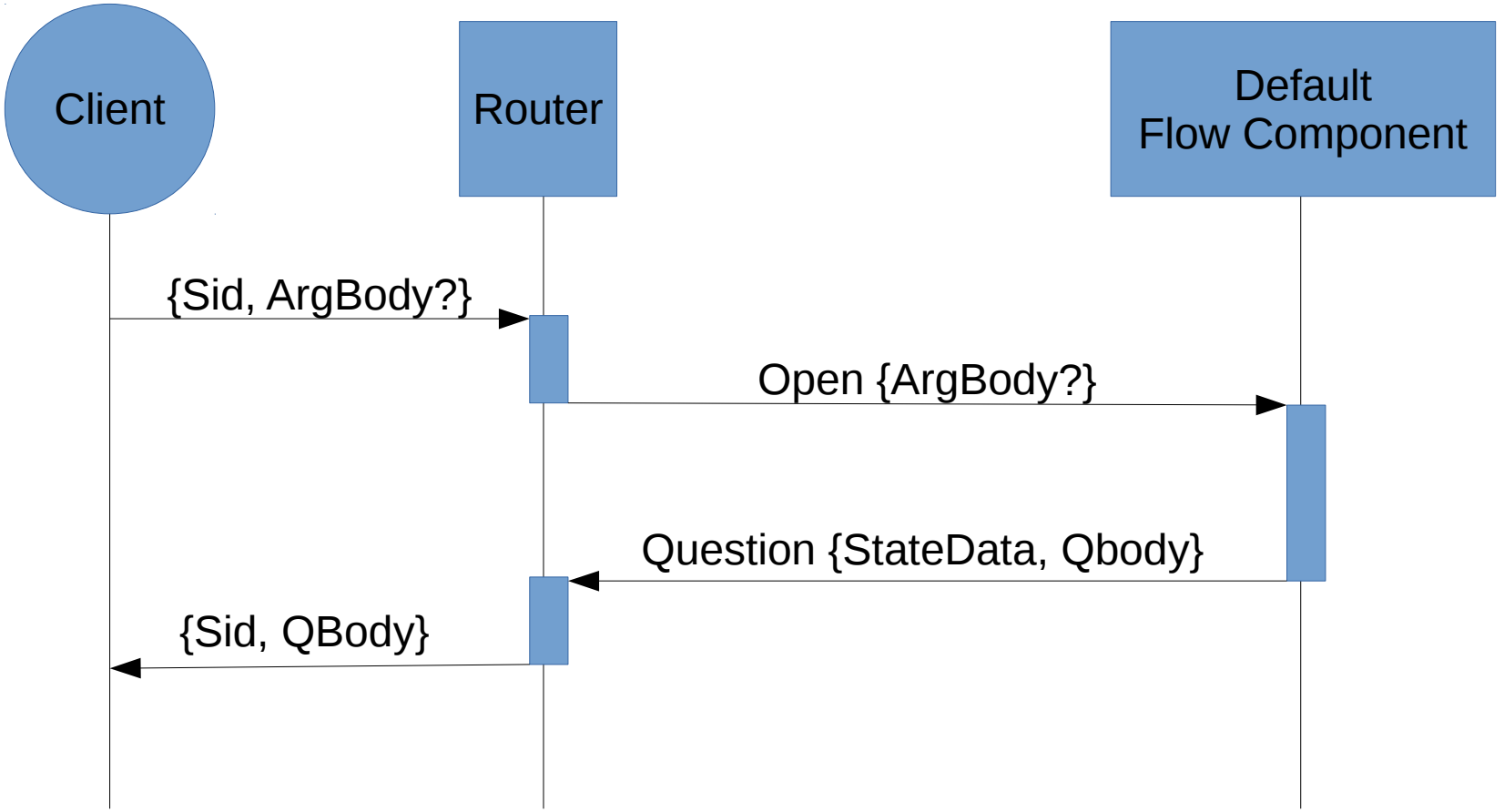
Стратегия
(стр. 2)

- Взаимодействие между компонентами, а так-же между компонентами и клиентом осуществляется через **Router**
- Один из компонентов объявляется корневым, от него начинает строиться вся иерархия
- Router хранит состояние сессии клиента в стеке
- Данные
  - состояния компонента в стеке
  - передаваемые при взаимодействии между компонентом и клиентом
  - передаваемые при взаимодействии между компонентами

  для Router обезличены и представляют собой **byte[]**
- Конфигурация Router декларативная
- Система самодокументируемая (Список компонентов, контракты, форматы, взаимосвязи в реальном времени)

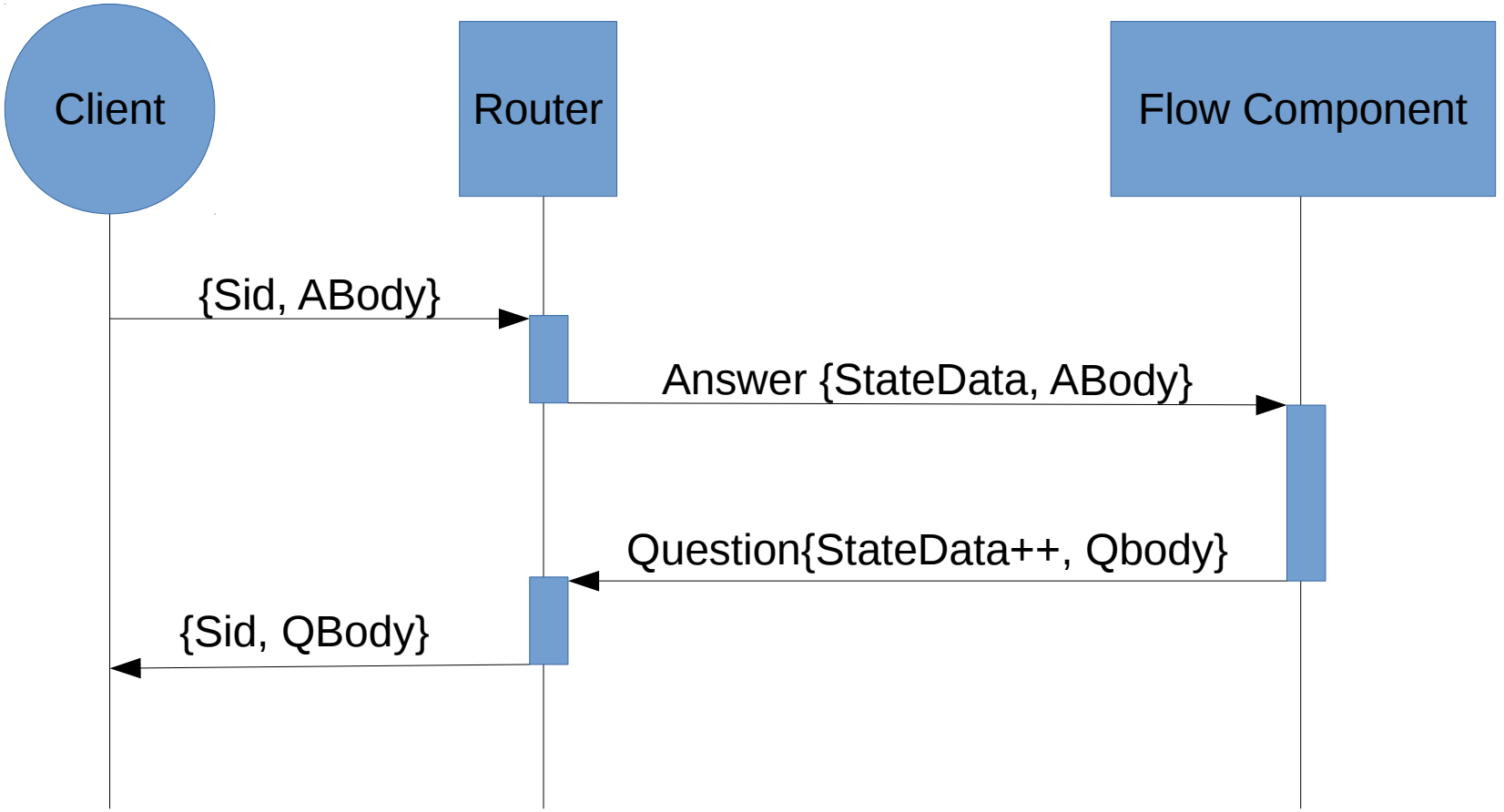
CloudBunch

Схема 1, первичный вход, нет данных о состоянии



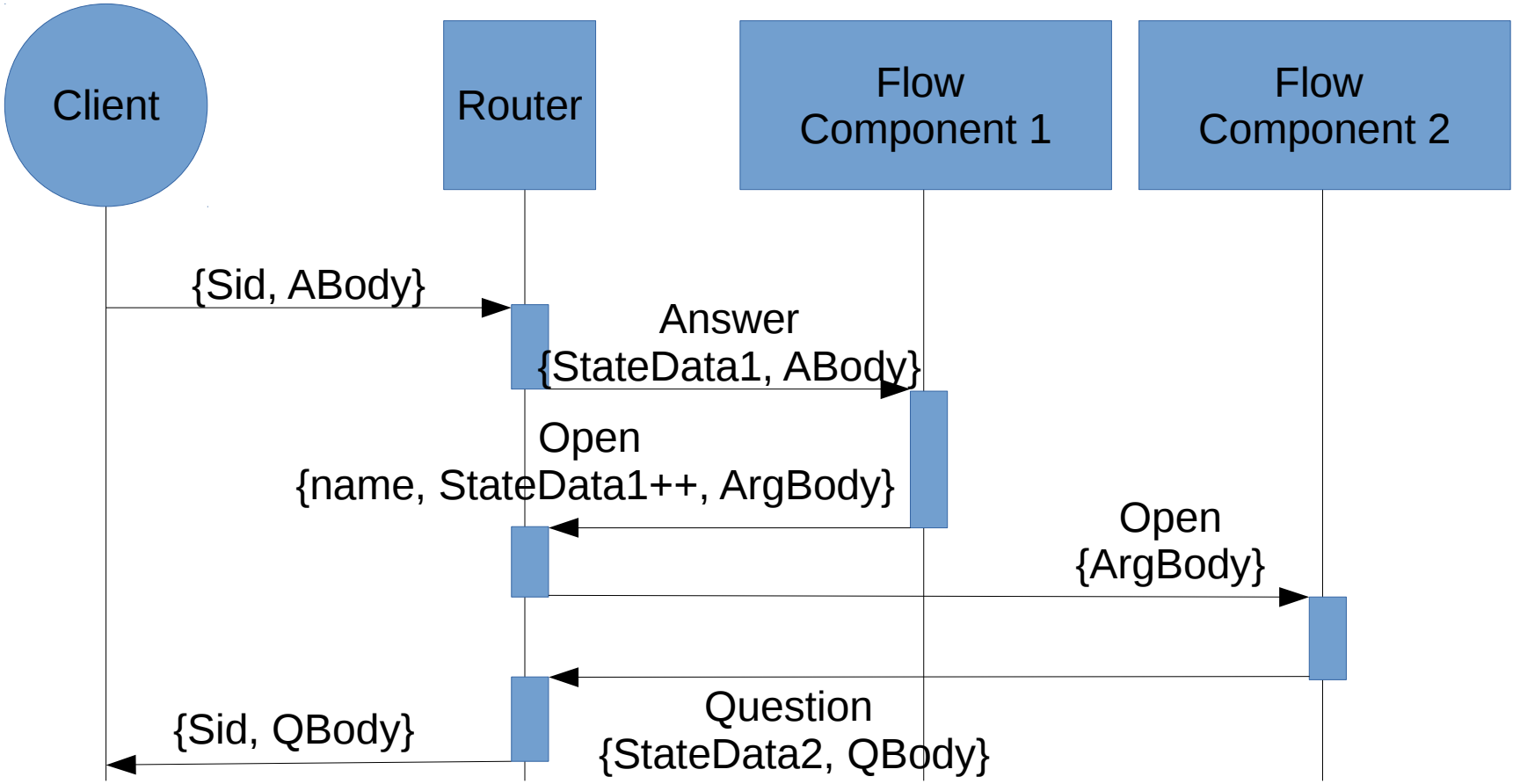
CloudBunch

Схема 2, взаимодействие с клиентом



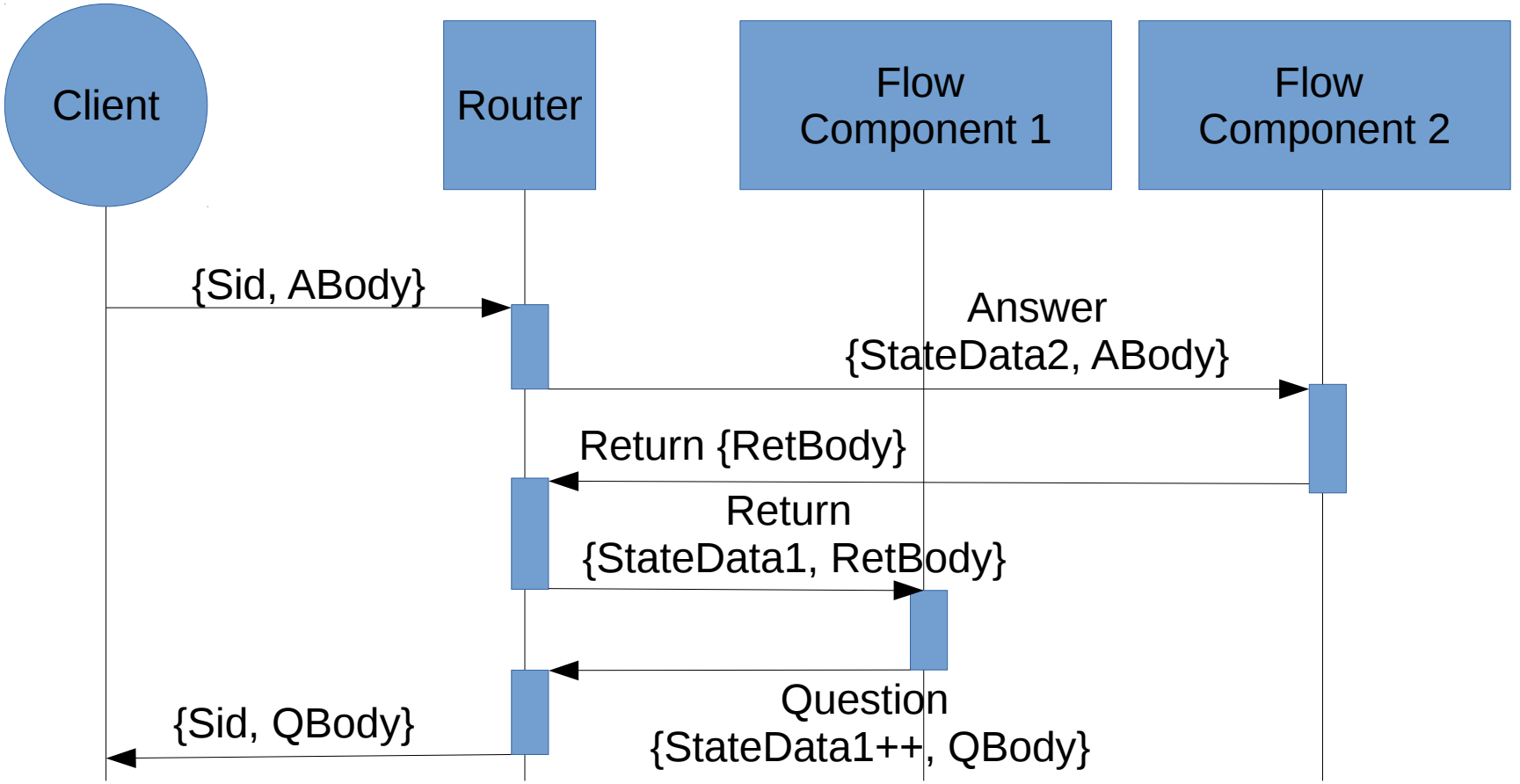
CloudBunch

Схема 3, вызов другого компонента



CloudBunch

Схема 4, возвращение к вызывавшему компоненту



Текущее состояние разработки

- Разработан контракт взаимодействия между Router и Flow компонентом
- Разработан рабочий прототип Router
  - Класс Router, конфигурация
  - Интерфейсы транспорта и хранилища сессий
  - Стэк и элемент стэка
- Разработан рабочий прототип Flow компонента
  - Абстрактный Flow
  - Абстрактный State и несколько его подтипов
  - Класс Contract, - декларация контрактов взаимодействия Flow компонентов между собой и с клиентом
- На базе прототипа Flow компонента разработан и внедряется в настоящий момент в банке LiS – фрэймворк для построения композитных транзакционных сервисов
- Разработана референсная реализация сервера Router