Beyond auto-scaling: application-aware optimal elasticity

Microservice 2019 (Dortmund, Germany)
19-20-21 February 2019

Iacopo Talevi - University of Bologna, Italy

joint work with:

Gianluigi Zavattaro - University of Bologna/INRIA, Italy
Jacopo Mauro - Southern Denmark University, Denmark

State-of-the-art deployment technologies

Are based on:

- autoscaling metrics

```
spec:
    ....
    minReplicas: 1
    maxReplicas: 10
    metrics:
    -   type: Resource
        resource:
            name: cpu
                target:
                type: Utilization
                averageUtilization: 50
```

Amazon EC2 Auto Scaling

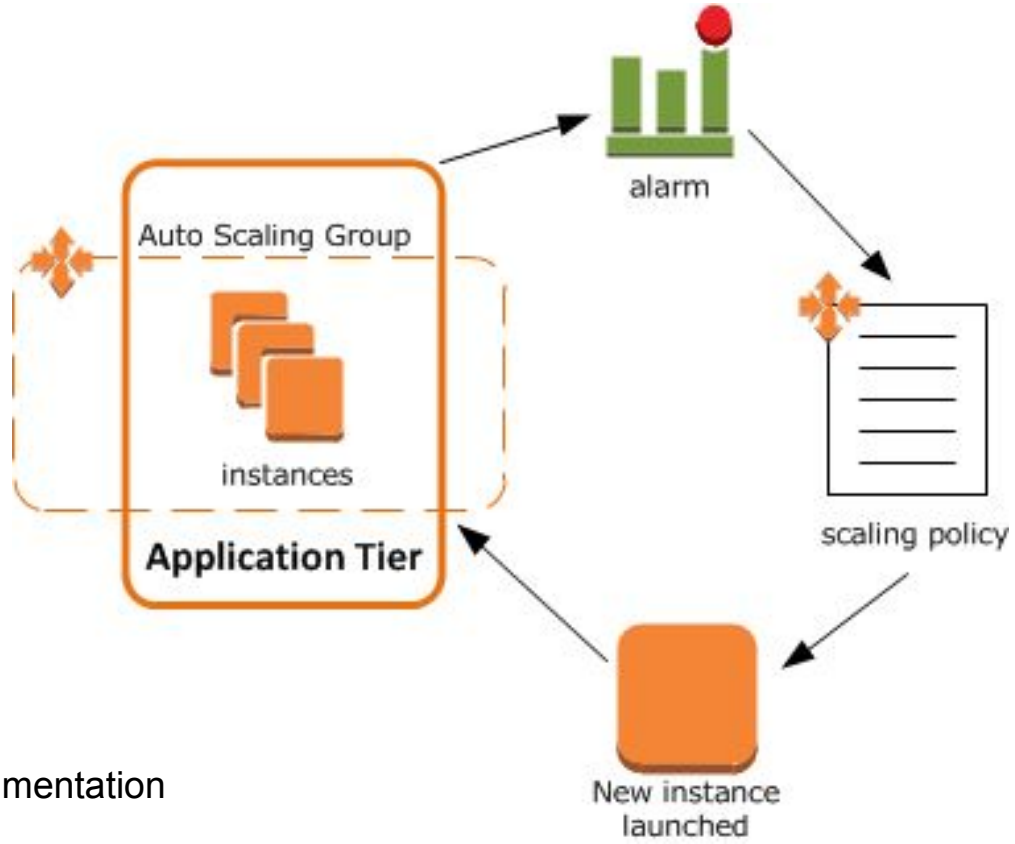
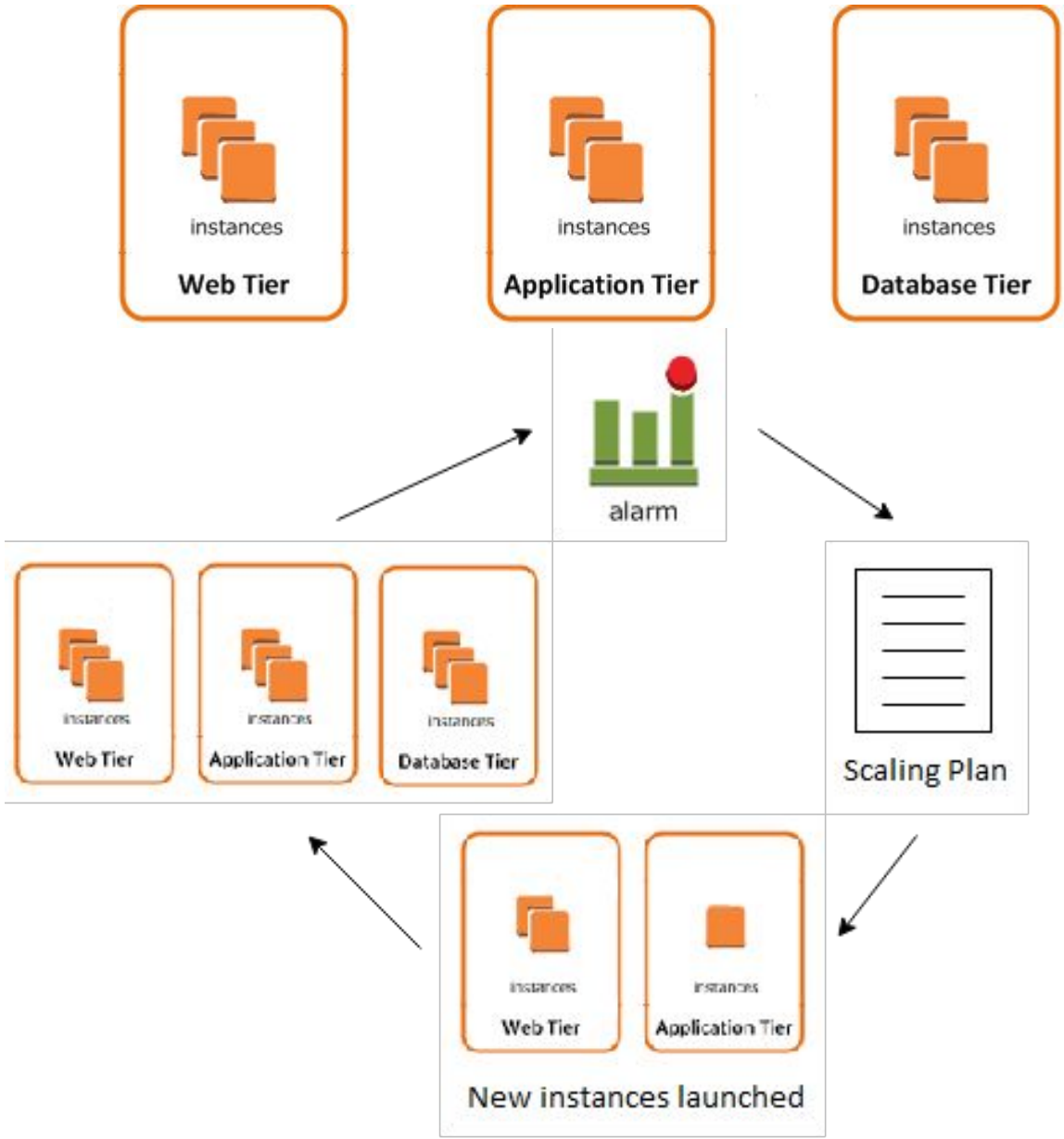


Image from: AWS Documentation

New Approach



Differences

Autoscaling	**Approach Proposed**
Reasoning at the local microservice level	Reasoning at the **global** architectural level
Focus on single microservice	Exploit **information on dependencies** among the microservices
Add/Remove several instances of single service	Add/Remove several instances of **different services**

Theorem Proved

The problem of deploying component-based software systems, in the **general case**, **is undecidable**.

(Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, and Gianluigi Zavattaro. Aeolus: A component model for the cloud. Inf. Comput. 2014)

We demonstrated that the **optimal deployment problem for microservices is decidable**.

(Mario Bravetti, Saverio Giallorenzo, Jacopo Mauro, Iacopo Talevi, and Gianluigi Zavattaro. Optimal and Automated Deployment for Microservices. In FASE, 2019)

What is necessary?

1. microservice **interdependencies**
2. **resource** consumption
3. define **“offline”** the **deployment goal**

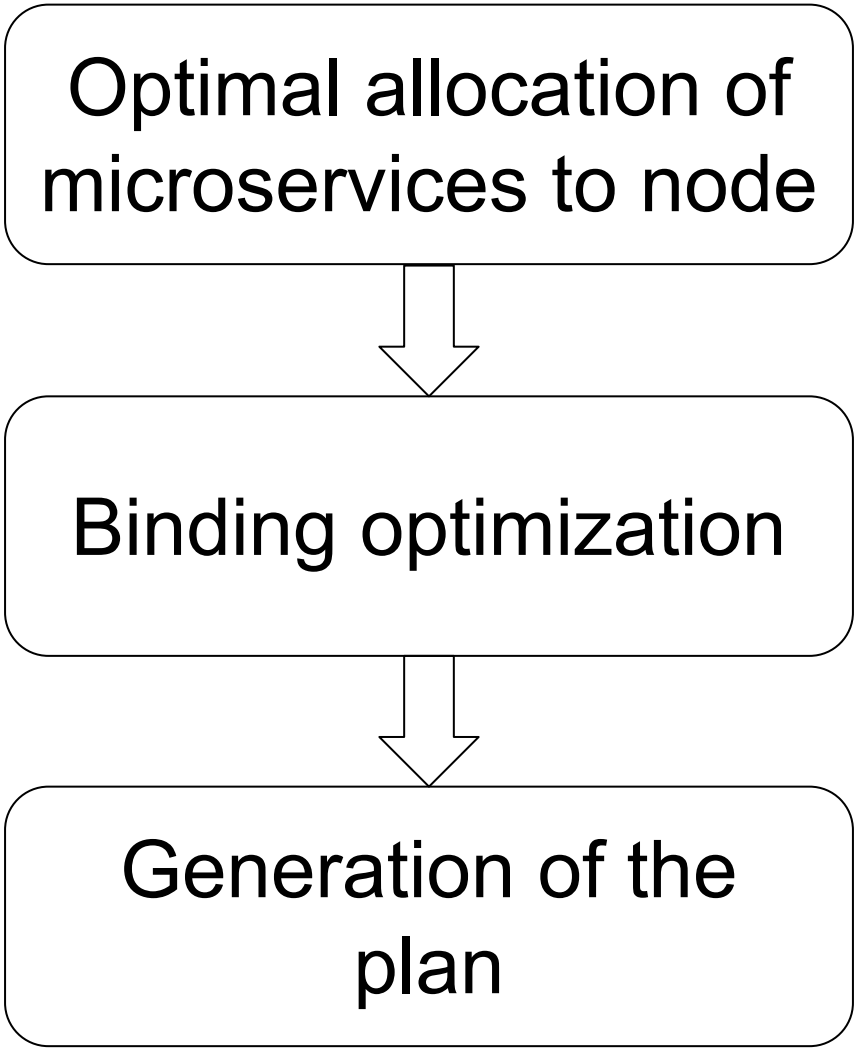
What means Optimal?

The **optimal deployment** has two properties:

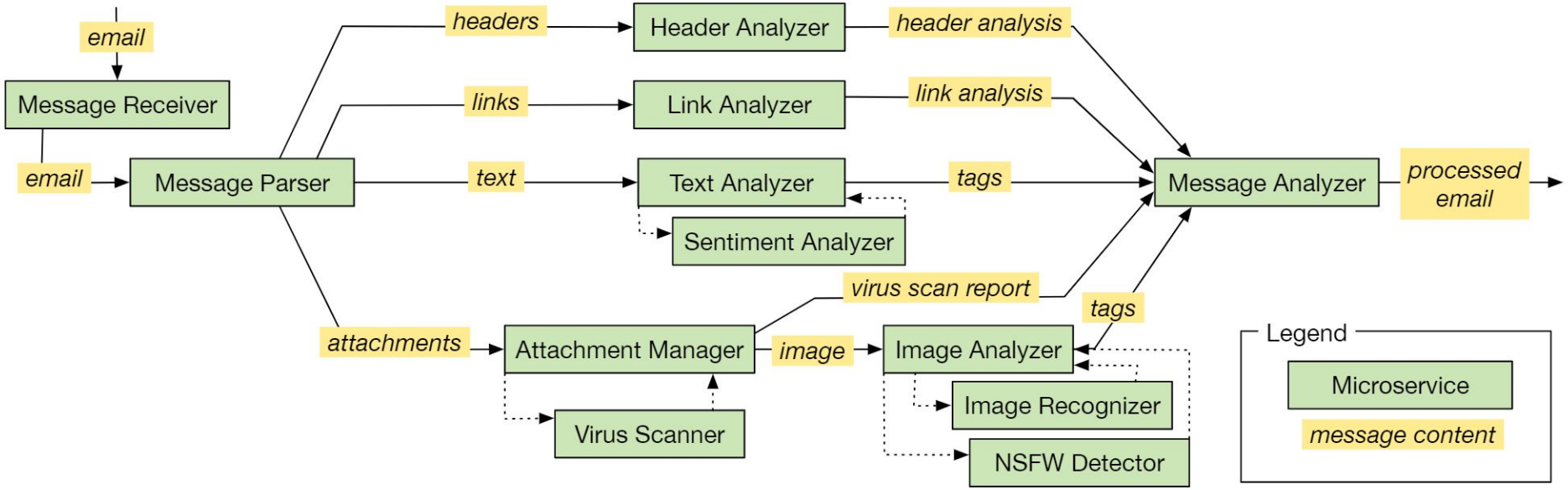
1. each used node has at least as many resources as those needed by the hosted microservices

2. the **total cost** (i.e., the sum of the costs) of the used nodes **is minimal**

Algorithm


Optimal allocation of microservices to node
Binding optimization
Generation of the plan

Case Study


email
Message Receiver
email
Message Parser
headers
Header Analyzer
header analysis
links
Link Analyzer
link analysis
text
Text Analyzer
tags
Sentiment Analyzer
Message Analyzer
processed email
attachments
Attachment Manager
virus scan report
tags
image
Image Analyzer
Virus Scanner
Image Recognizer
NSFW Detector
Legend
Microservice
message content

Tools Used

1. **Abstract Behavioral Specification (ABS) language**

2. **SmartDepl**, an ABS extension that allows to specify:
   a. the **resources provided** by the deployment components
   b. ABS Classes annotations that indicate the computing **resources necessary** and **dependencies**
   c. the specification of **declarative deployment rules**

   It uses an external solver that generates ABS classes modeling the optimal deployments (**Zephyurs2**)

Necessary Annotations

For each node type:

```
"c4_large" : {
      "cost" : 119,
      "payment_interval" : 1,
      "resources" : {
            "Cores" : 2,
            "Memory" : 375
      }
}
```

Necessary Annotations

For each microservices:

```
"class" : "MessageReceiver",
"scenarios" : [
    {
        ...
        "cost" : {
            "Cores" : 2,
            "Memory" : 200
        },
        "sig" : [
            {
                "kind" : "require",
                "type" : "MessageParser_LoadBalancerInterface"
            }
        ],
        ...
    }
]
```

Necessary Annotations

For each Load Balancer:

```
"class" : "MessageParser_LoadBalancer",
"scenarios" : [
    {
        ….
        "cost" : { "Cores" : 2,  "Memory" : 200 },
        ….
        "methods" : [ {
            "add" : {
                "name" : "connectInstance",
                "param_type" : "MessageParserInterface"
            },
            "remove" : {
                "name" : "disconnectInstance",
                "return_type": "MessageParserInterface"
            }
        } ]
    }
]
```

Necessary Annotations

For each deployment plan:

```
"id":"MainSmartDeployer",
"specification":"       MessageReceiver = 1 and
                        MessageParser = 1 and
                        HeaderAnalyser = 1 and
                        LinkAnalyser = 1 and ….”
...
"cloud_provider_DC_availability":{
      "c4_large":40,
      "c4_xlarge":40,
      "c4_2xlarge":40
},
"bind preferences":[
      "local",
      "sum ?x of type MessageParser in '.*' :  forall ?y of type
MessageParser_LoadBalancer in '.*' : ?x used by ?y", ….
],
….
```

Deployment Plans Created

Microservice (max computational load)	Initial (10K)	+20K	+50K	+80K
MessageReceiver(∞)	1	-	-	-
MessageParser(40K)	1	-	+1	-
HeaderAnalyzer(40K)	1	-	+1	-
LinkAnalyzer(40K)	1	-	+1	-
TextAnalyzer(15K)	1	+1	+2	+2
SentimentAnalyzer(15K)	1	+3	+4	+6
AttachmentsManager(30K)	1	+1	+2	+2
VirusScanner(13K)	1	+3	+4	+6
ImageAnalyzer(30K)	1	+1	+2	+2
NSFWDetector(13K)	1	+3	+4	+6
ImageRecognizer(13K)	1	+3	+4	+6
MessageAnalyzer(70K)	1	+1	+2	+2

The ABS code modeling the system and the generated code are publicly available at:
https://github.com/IacopoTalevi/SmartDeploy-ABS-ExampleCode

Conclusion

- In this work, we prove that the generation of a deployment plan for an architecture of microservices **is decidable and fully automatable**, testing it in a real-world microservice architecture model

- Regarding autoscaling, our work is an example of how we can go **beyond single-component horizontal scaling** policies

As future work, we want to study the possibilities of **speed-up** the solution of the optimization problems to obtain **on-the-fly deployment plans**.