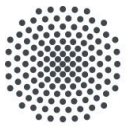
Serverless or Serverful? A Pattern-based Approach for Exploring Hosting Alternatives



University of Stuttgart

Vladimir Yussupov[1], Uwe Breitenbücher[1], Antonio Brogi[2], Lukas Harzenetter[1], Frank Leymann[1], and Jacopo Soldani[2]

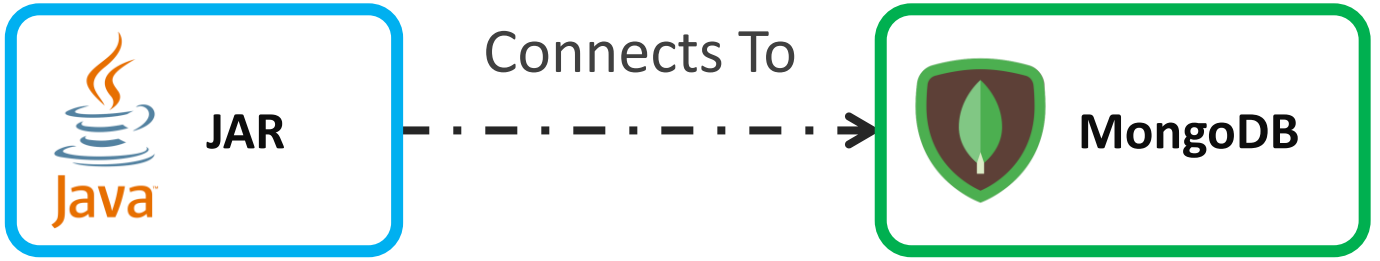
[1] *University of Stuttgart, [lastname]@iaas.uni-stuttgart.de*

[2] *University of Pisa, [name.lastname]@unipi.it*

Motivational Example: Cloudius and the Hosting Conundrum

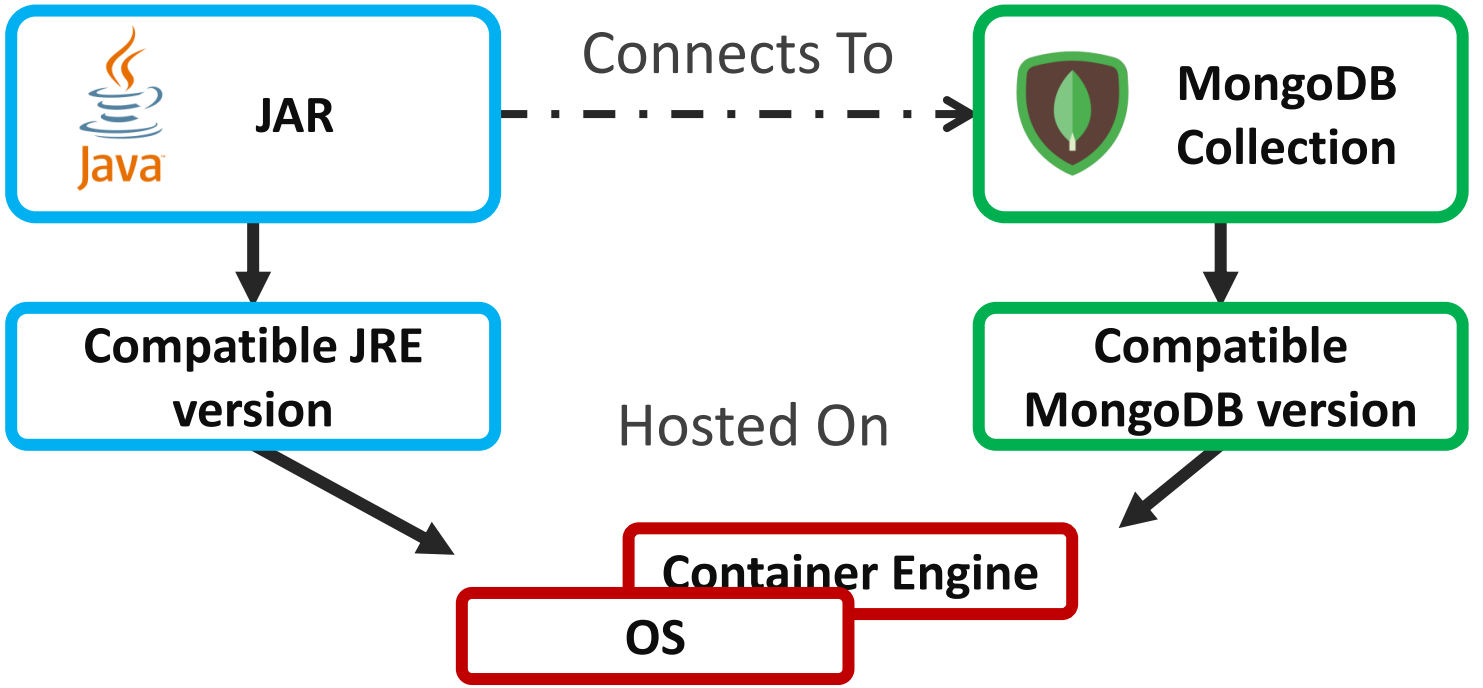
- Cloudius learns about SummerSoC 2022 happening on-site ☺
- “I should capture and organize the sessions and presentations I want to attend”

Motivational Example: Cloudius and the Hosting Conundrum



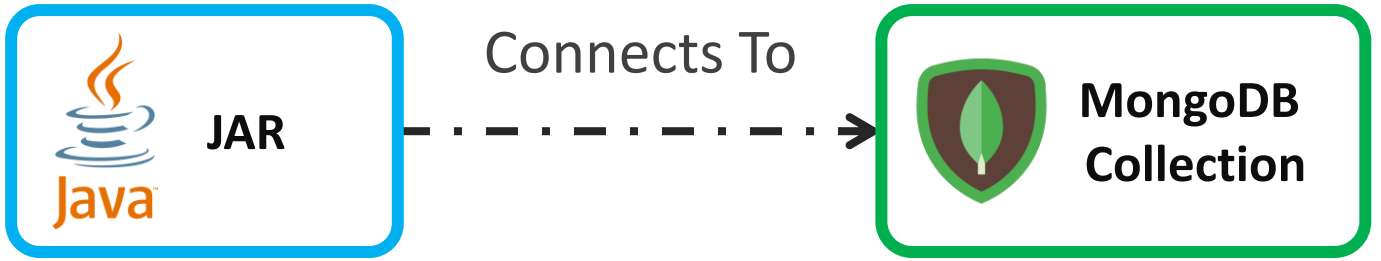
- Let's make a ToDo list app to keep track of the TODOs!
- The chosen technologies are
  - Java v11+
  - MongoDB v4.4+

Motivational Example: Cloudius and the Hosting Conundrum



- JAR + Java runtime
- MongoDB Collection + MongoDB
- Container orchestrator, Operating system, …

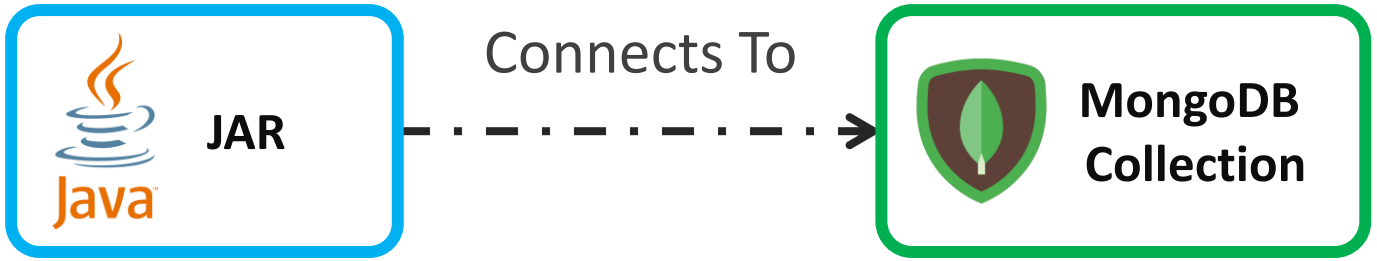
Motivational Example: Cloudius and the Hosting Conundrum



Hmm, How Should I Host It?

Let's ask the magic crystal!

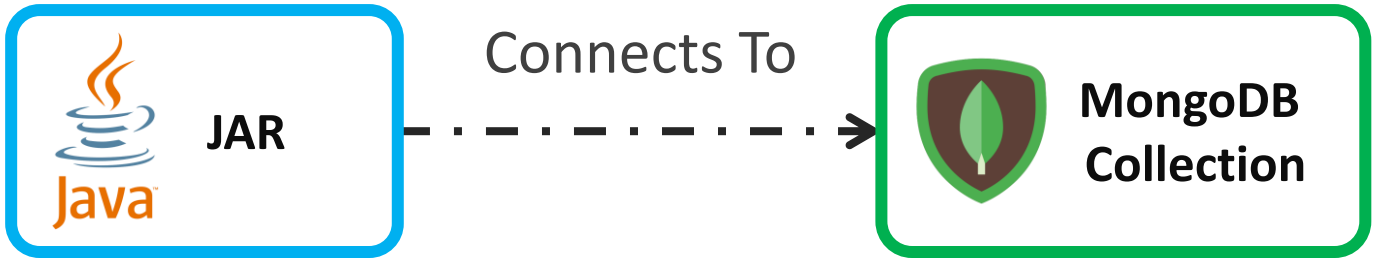
Motivational Example: Cloudius and the Hosting Conundrum



Cloudius@MagicCrystal: ”Searching for a hosting option…”

- On-premises
- Bare metal cloud offerings, Infrastructure-as-a-Service
- Container-as-a-Service, Platform-as-a-Service, Function-as-a-Service , Database-as-a-Service
- …

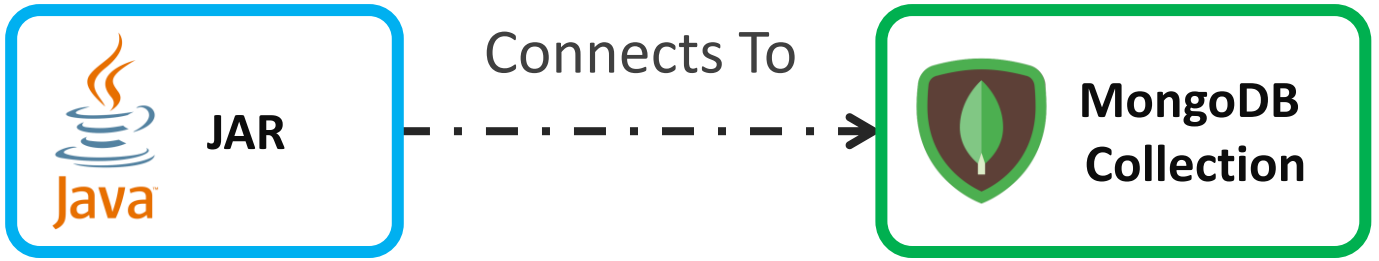
Motivational Example: Cloudius and the Hosting Conundrum



Cloudius@MagicCrystal: ”Analyzing management efforts”:

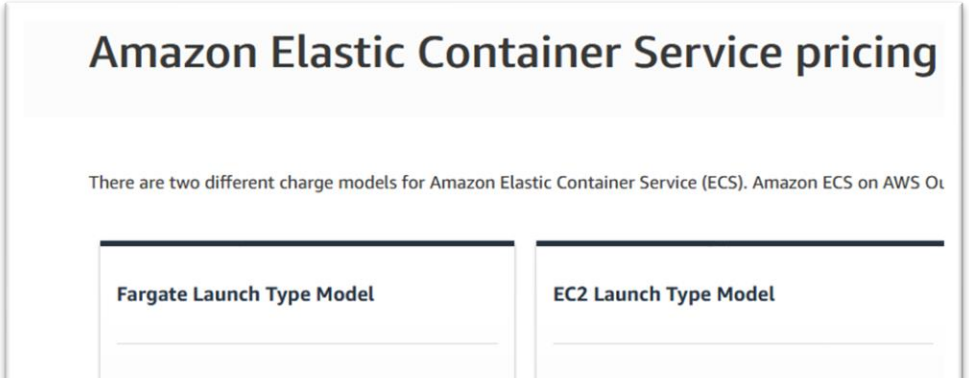
- Which general-purpose components are preferred?
- Who sets these general-purpose components up?
- Who is responsible for specifying infrastructure resources and scaling rules?

Motivational Example: Cloudius and the Hosting Conundrum

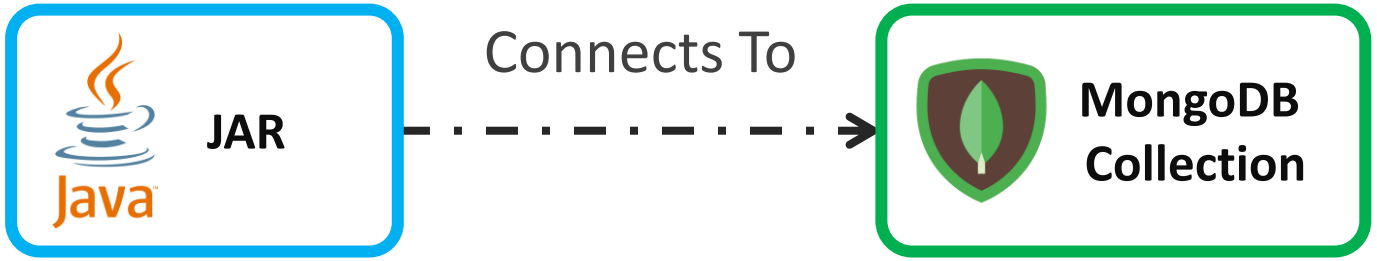


Cloudius@MagicCrystal: "… management requirements may differ for the same cloud service model"

- PaaS / CaaS with consumer- or provider-managed scaling configuration / resources allocation



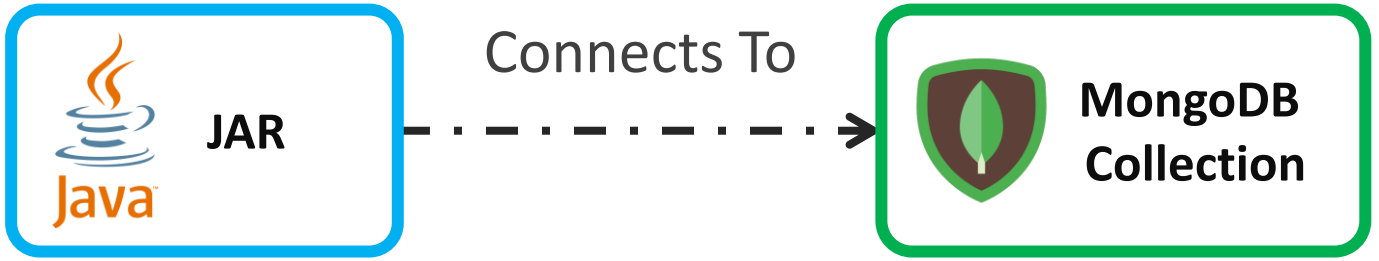
Motivational Example: Cloudius and the Hosting Conundrum



Cloudius@MagicCrystal: ”… management efforts may be similar for different cloud service models”

- Deployment stack management for bare-metal cloud and IaaS
- Scaling configuration for certain CaaS and FaaS, e.g., AWS Fargate and AWS Lambda

Motivational Example: Cloudius and the Hosting Conundrum

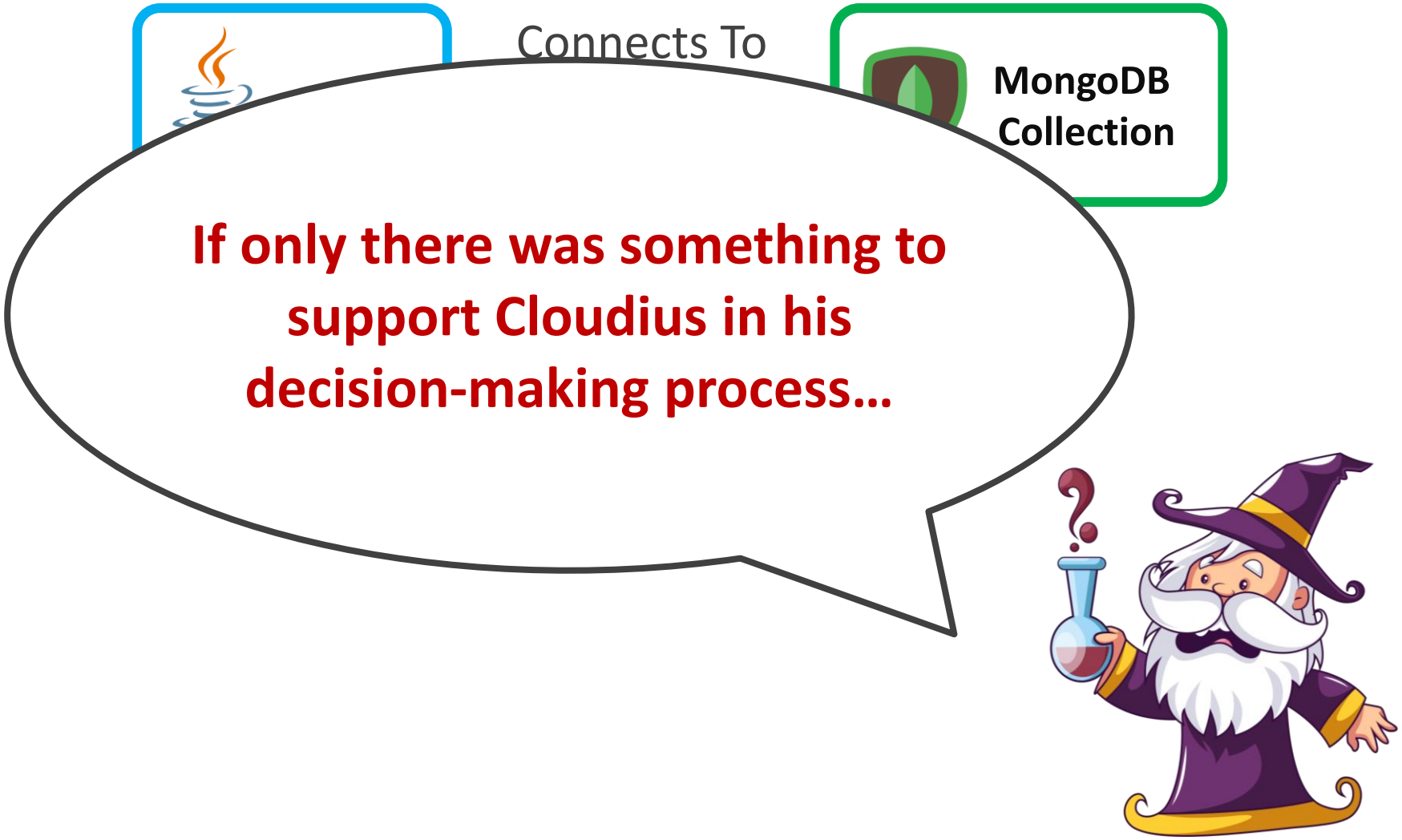


Magic Crystal:

500

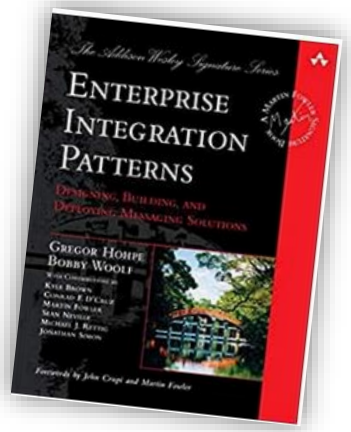
Internal Server Error

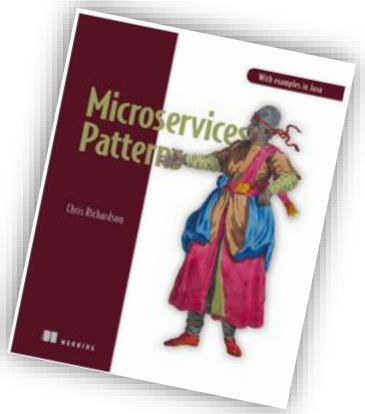
Motivational Example: Cloudius and the Hosting Conundrum



Patterns & Design Decisions

- Describe proven solutions for frequently reoccurring problems
- Documented in an abstract way and typically follow a well-defined structure
- Patterns are not invented, they are found
- Simplify the architectural decision making





Pattern's Description Examples

Block Storage

Centralized storage is integrated into servers as a local hard drive managed by the operating system to enable access to this storage via the local file system.

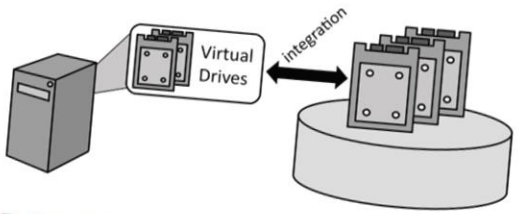
How can central storage be accessed as a local drive by servers and hosted applications?

Context

Virtual and non-virtualized servers offered as Infrastructure as a Service (IaaS) can be managed significantly easier if they do not store any state information locally, i.e., on their (virtual) hard drives. This eases their provisioning, decommissioning, and failure handling.

Solution

Centralized storage is accessed by servers as if it was a local hard drive, also referred to as block device.



Related Patterns

Environment-based Availability, Blob Storage, Strict Consistency, Eventual Consistency

Infrastructure as a Service (IaaS)

Providers share physical and virtual hardware IT resources between customers to enable self-service, rapid elasticity, and pay-per-use pricing.

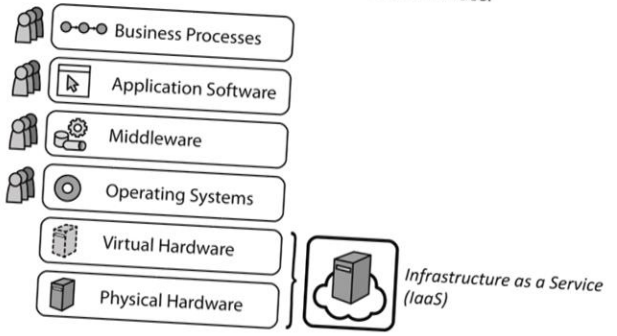
How can different customers share a physical hosting environment so that it can be used on-demand with a pay-per-use pricing model?

Context

In the scope of Periodic Workloads with reoccurring peaks and the special case of Once-in-a-lifetime Workloads with one dramatic increase in workload, IT resources have to be provisioned flexibly.

Solution

A provider offers physical and virtual hardware, such as servers, storage and networking infrastructure that can be provisioned and decommissioned quickly through a self-service interface.



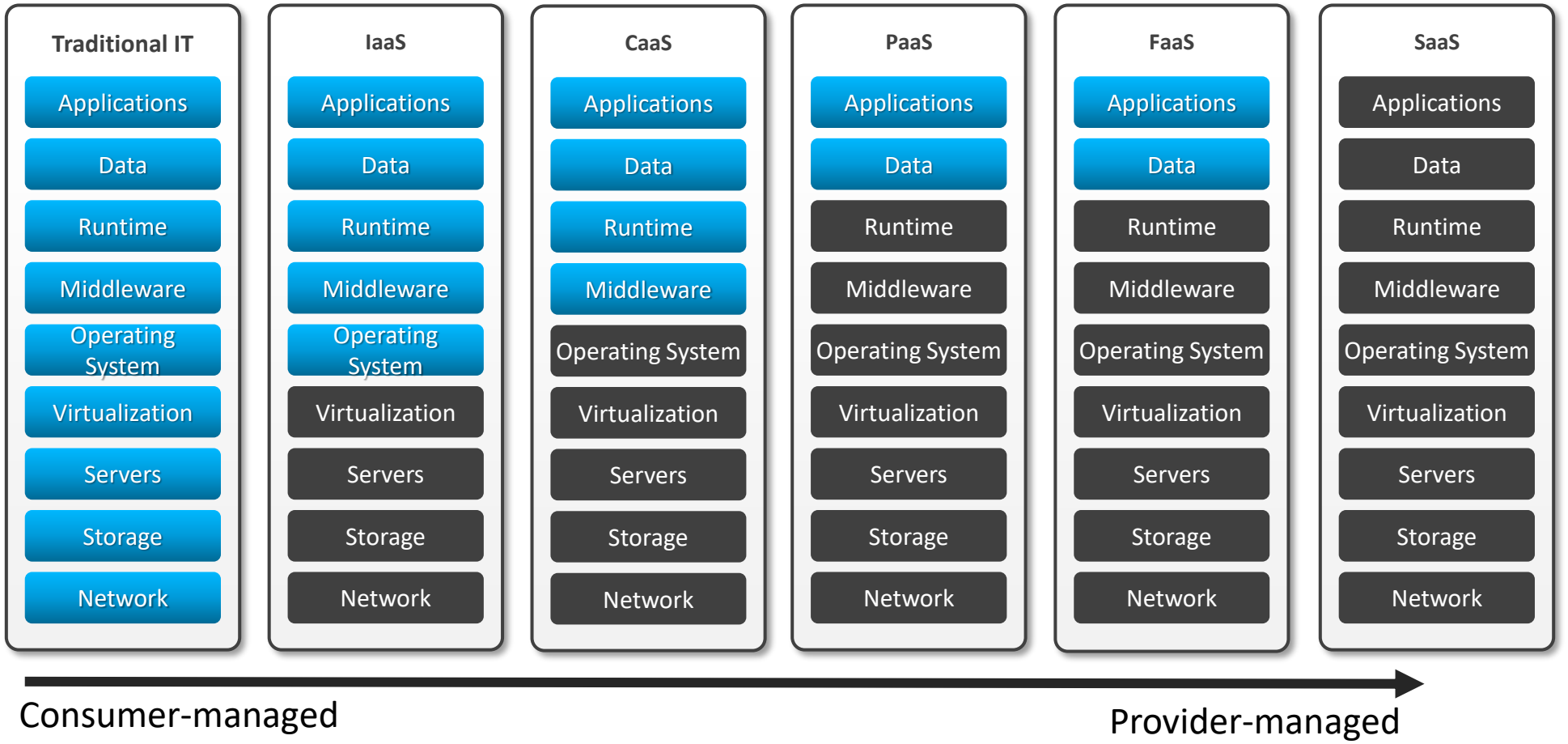
Related Patterns

Elastic Infrastructure, Hypervisor, Block Storage, Virtual Networking

Component Hosting and Management Pattern Language

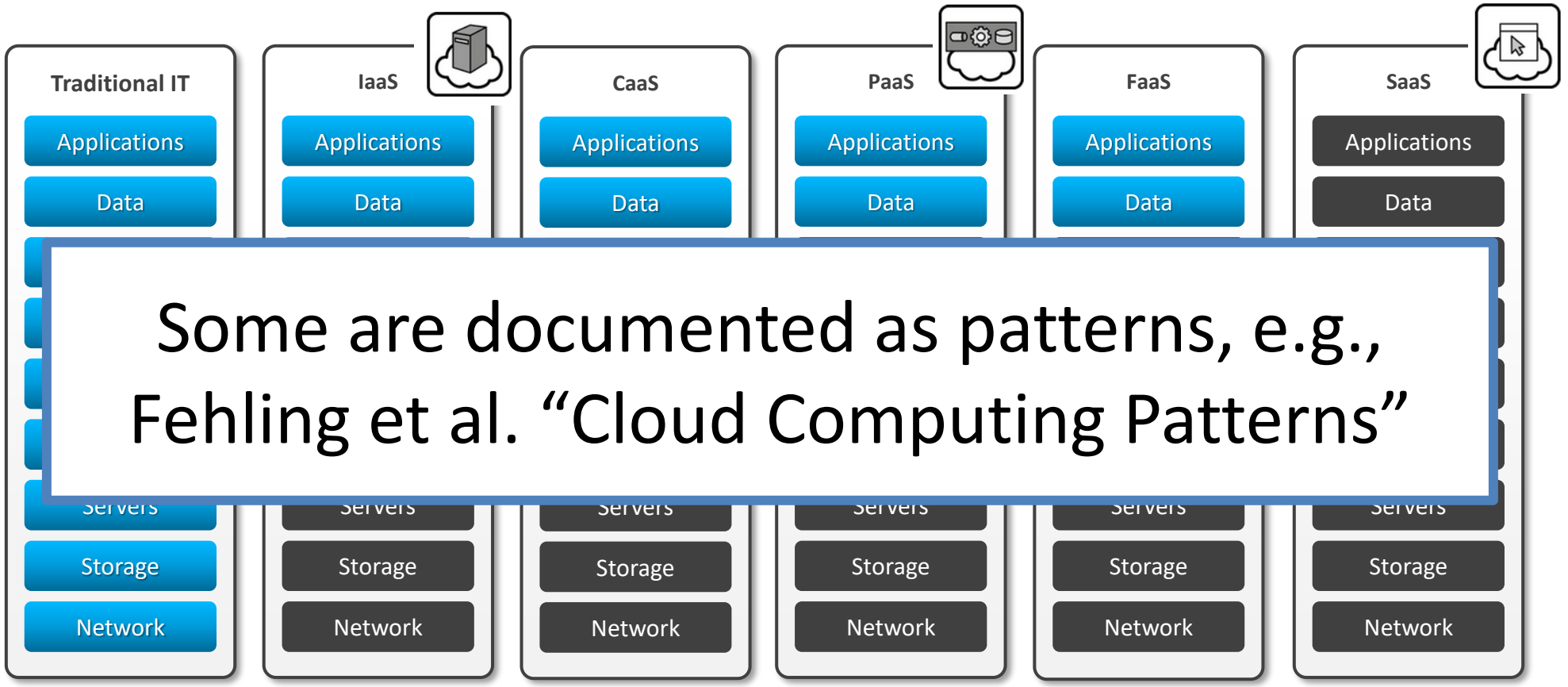
A Spectrum of Hosting and Management Decisions

Deployment stack management as in cloud service models:



Component Hosting Options in a Spectrum

Deployment stack management as in cloud service models:



Component Hosting Options in a Spectrum

BUT

- Different cloud service models may have significant overlaps, essentially describing a similar “hosting experience”
  - On-premises -> Bare metal cloud -> IaaS

Component Hosting Options in a Spectrum

BUT

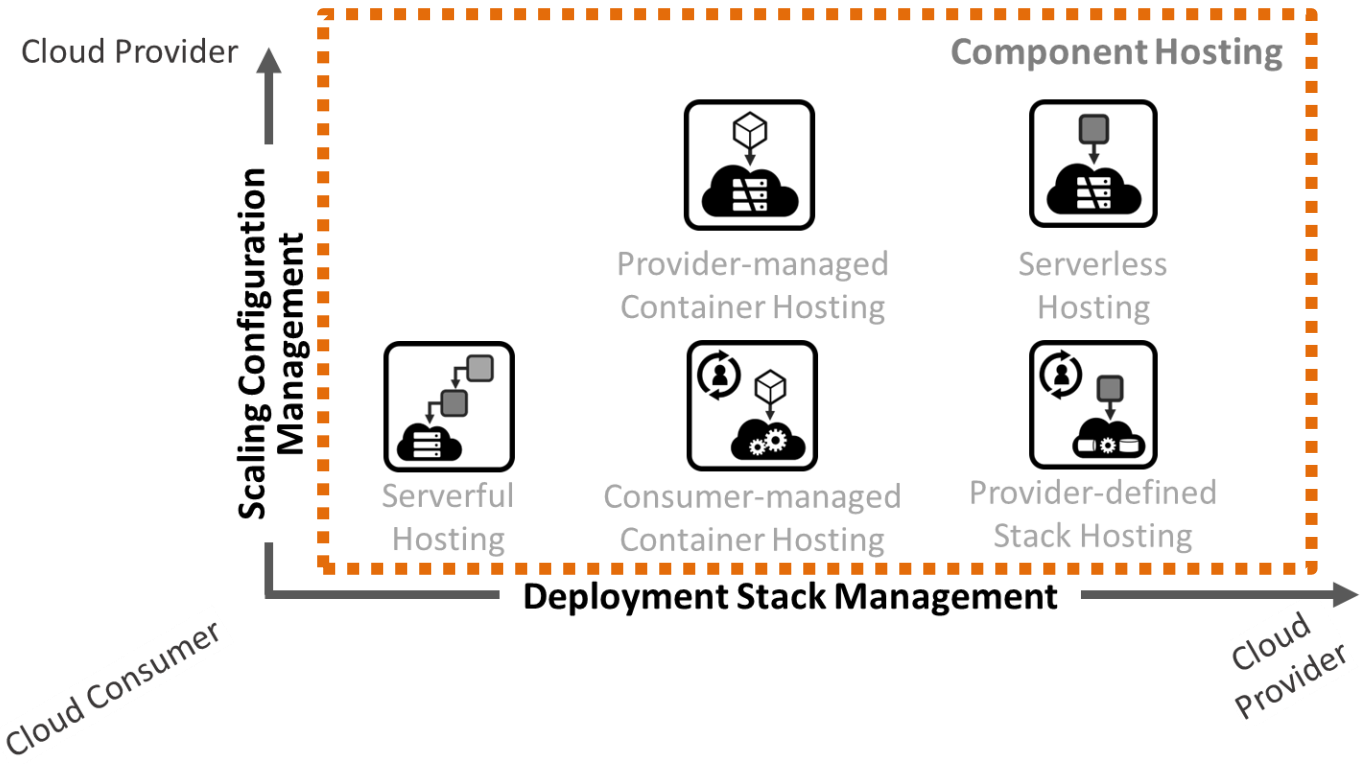
- Different cloud service models may have significant overlaps, essentially describing a close “hosting experience”
  - On-premises -> Bare metal cloud -> IaaS
- Scaling configuration requirements might be different for the same cloud service model
  - FaaS-like CaaS vs. more consumer-managed versions of CaaS
  - PaaS offerings vary in management efforts too, e.g., Heroku vs AWS Beanstalk

Component Hosting Options in a Spectrum

BUT

- Different cloud service models may have significant overlaps, essentially describing a close “hosting experience”
  - On-premises -> Bare metal cloud -> IaaS
- Scaling configuration requirements might be different for the same cloud service model
  - FaaS-like CaaS vs. more managed version of CaaS
  - PaaS offerings vary in management efforts too, e.g., Heroku vs AWS Beanstalk
- Terms like serverless are often used w.r.t. different cloud service models too
  - FaaS, DBaaS, SaaS, …

Component Hosting and Management Patterns: Previous Work



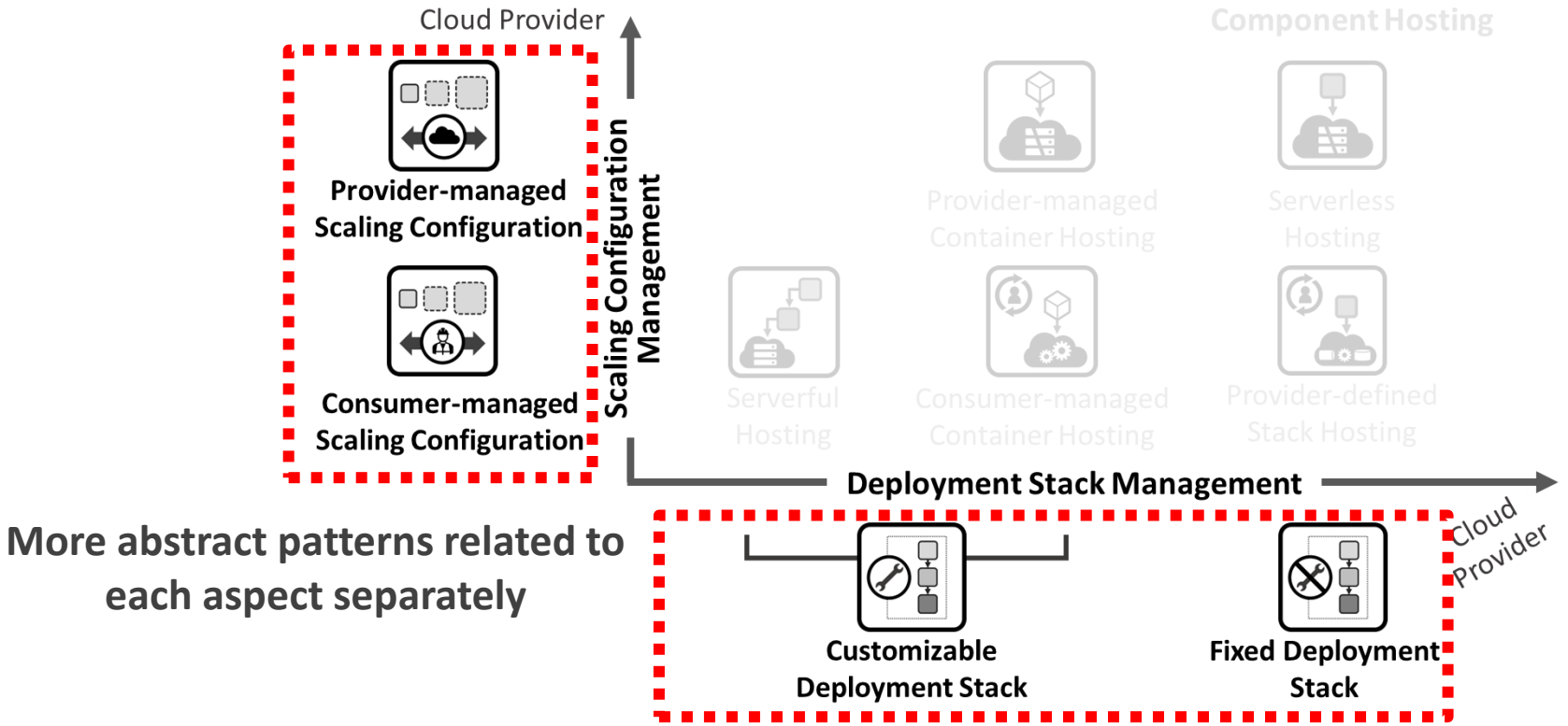
Documented in the previous work*, capture management trade-offs

Capture abstract and more concrete hosting solutions based on 2 aspects:

- **Deployment Stack Management**
- **Scaling Configuration Management**

* Yussupov, V., et al.: From Serverful to Serverless: A Spectrum of Patterns for Hosting Application Components. In: Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)

Component Hosting and Management Patterns: Extended



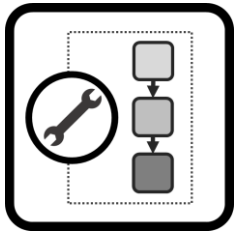
More abstract patterns related to each aspect separately

Capture abstract and more concrete hosting solutions based on 2 aspects:

- **Deployment Stack Management**
- **Scaling Configuration Management**

* Yussupov, V., et al.: From Serverful to Serverless: A Spectrum of Patterns for Hosting Application Components. In: Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)

Customizable Deployment Stack Pattern



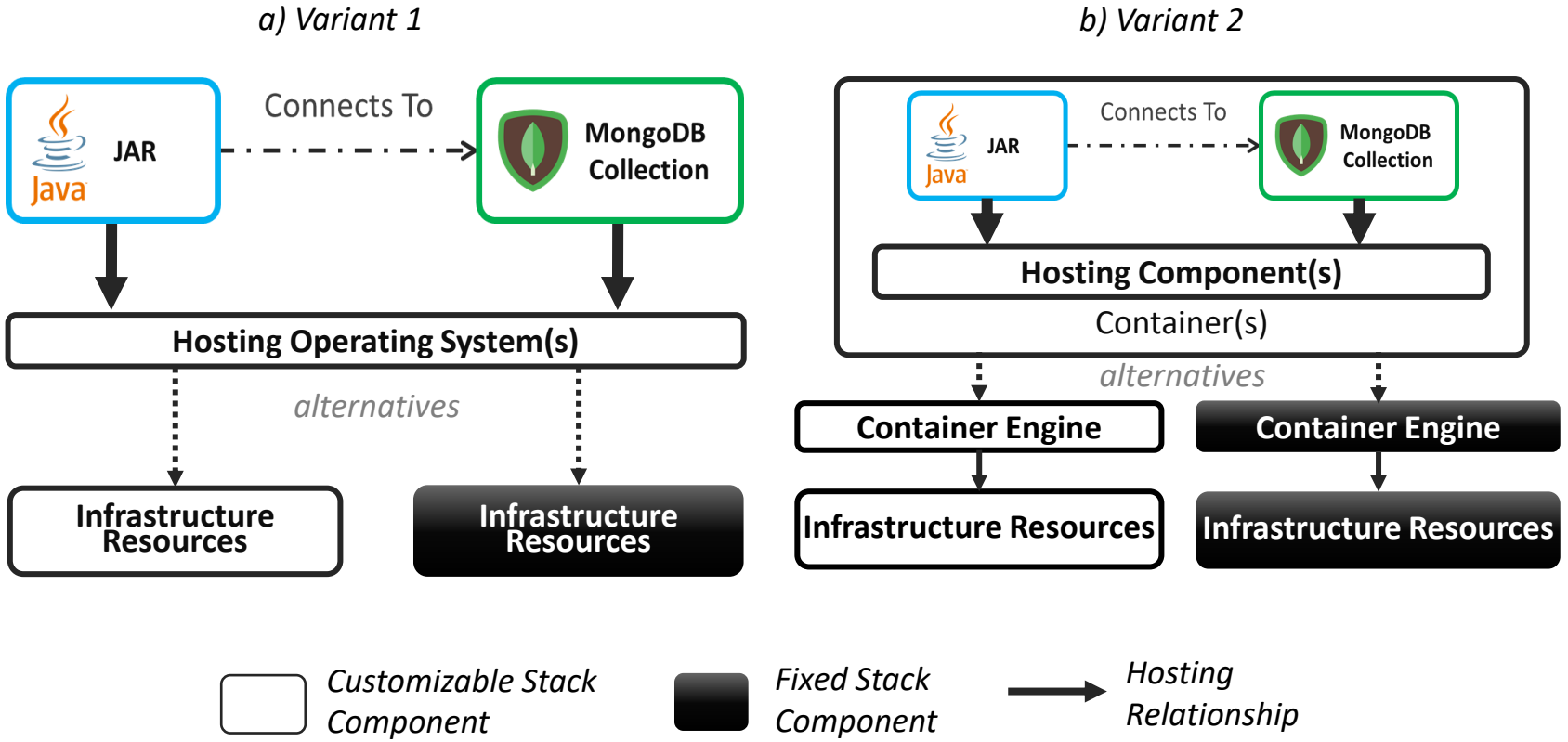
Problem: *How to host a software component when it requires customization of the underlying infrastructure or the host environment it is running on?*

Context: A software component needs to be hosted such that cloud consumers can customize the underlying deployment stack with possibly nested software layers by adding, removing, or changing components in it or configuring the stack in a special way.

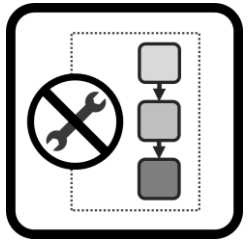
Solution: Host the component on a *Customizable Deployment Stack* where cloud consumers manage the required infrastructure, execution environment, and middleware components. Hence, it is possible to install and configure hosting components with required custom dependencies.

Known Uses: AWS EC2, Azure IaaS, IBM Cloud Bare Metal Servers, Azure Kubernetes Service, AWS Fargate, Google CloudRun

Customizable Deployment Stack: Example



Fixed Deployment Stack Pattern



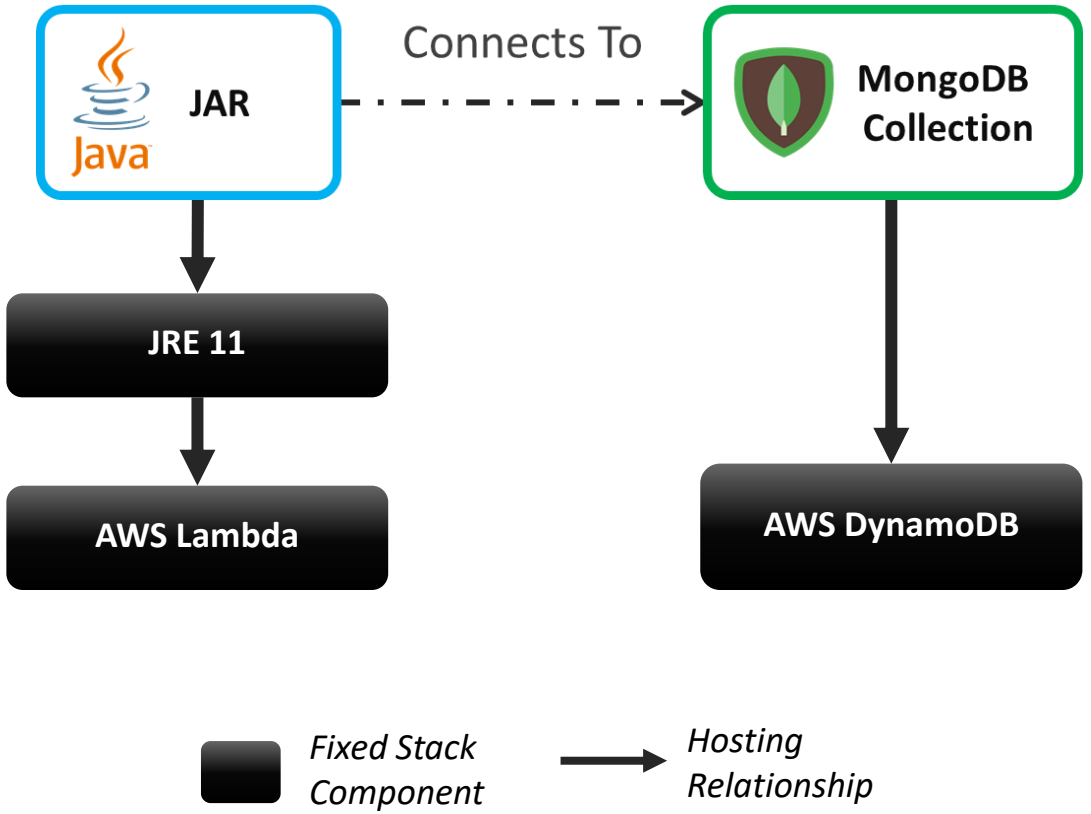
Problem: *How to host a software component that requires no special underlying infrastructure or customization of the host environment it is running on?*

Context: A software component needs to be hosted on a deployment stack, but there is no need for special customization of the stack by adding, removing, or changing components in it or configuring it in a special way.

Solution: Host the component on a *Fixed Deployment Stack* for which cloud providers set up, configure, and maintain the required infrastructure, execution environment, and middleware components. Hence, consumers can directly host software components on chosen provider-defined stacks without the need to configure or customize the underlying components.

Known Uses: AWS Beanstalk, Azure App Service, IBM Cloud Functions, AWS S3, IBM Cloud Databases for Redis

Fixed Deployment Stack: Example



Consumer-managed Scaling Configuration Pattern

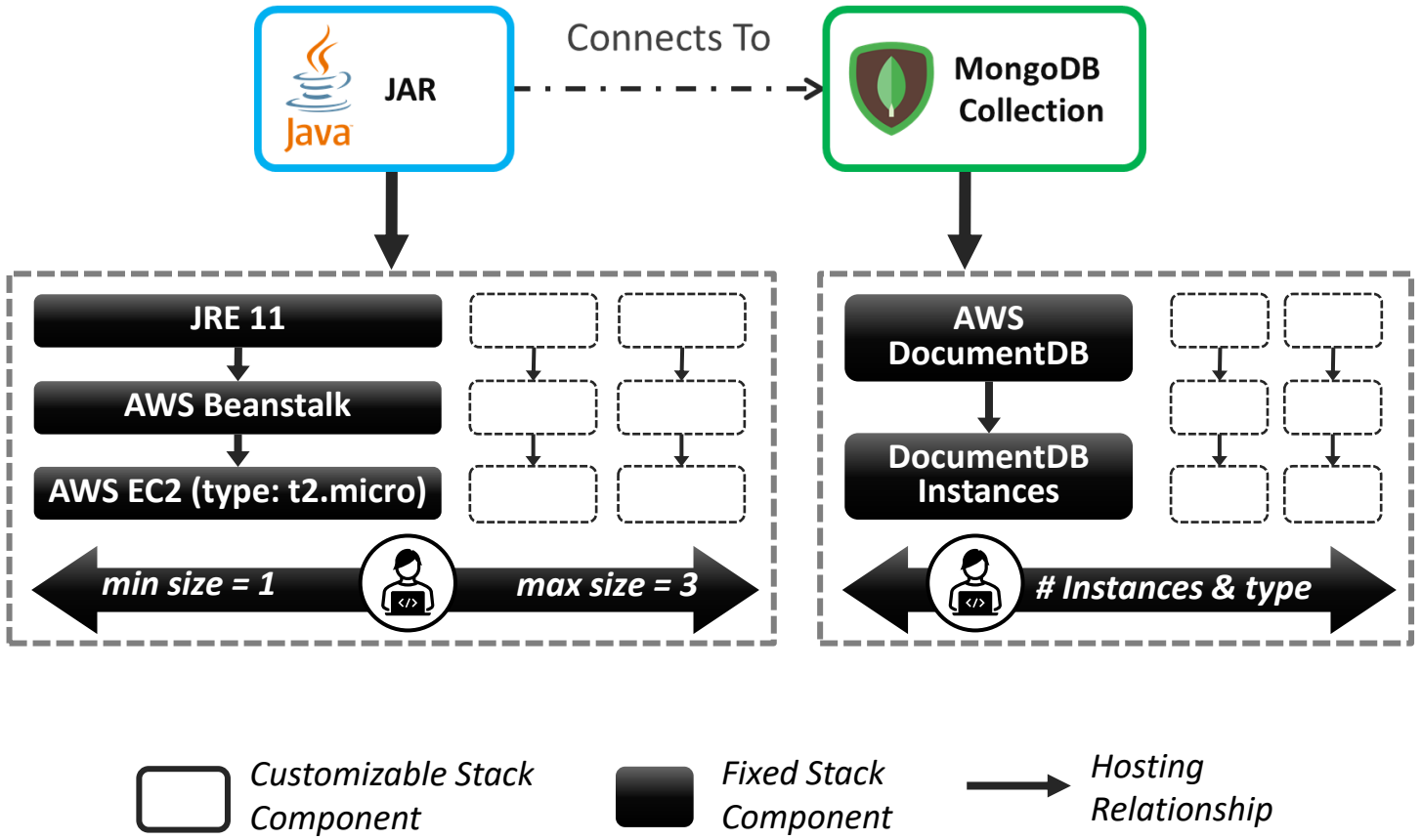
Problem: *How to host a software component that needs to be scaled horizontally while also requiring a tailored scaling configuration?*

Context: A software component needs to be hosted on a deployment stack for which cloud consumers can explicitly specify the underlying infrastructure resources and retain a high level of control over the horizontal scaling rules, e.g., the size of virtual machines cluster and the desired autoscaling rules for it.

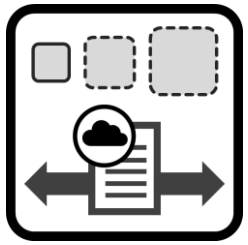
Solution: Host the component on a deployment stack supporting *Consumer-Managed Scaling Configuration*, i.e., consumer-specified infrastructure resources and horizontal scaling rules.

Known Uses: AWS EC2, Azure IaaS, IBM Cloud Bare Metal Servers, Azure Kubernetes Service, AWS Beanstalk

Consumer-managed Scaling Configuration: Example



Provider-managed Scaling Configuration Pattern



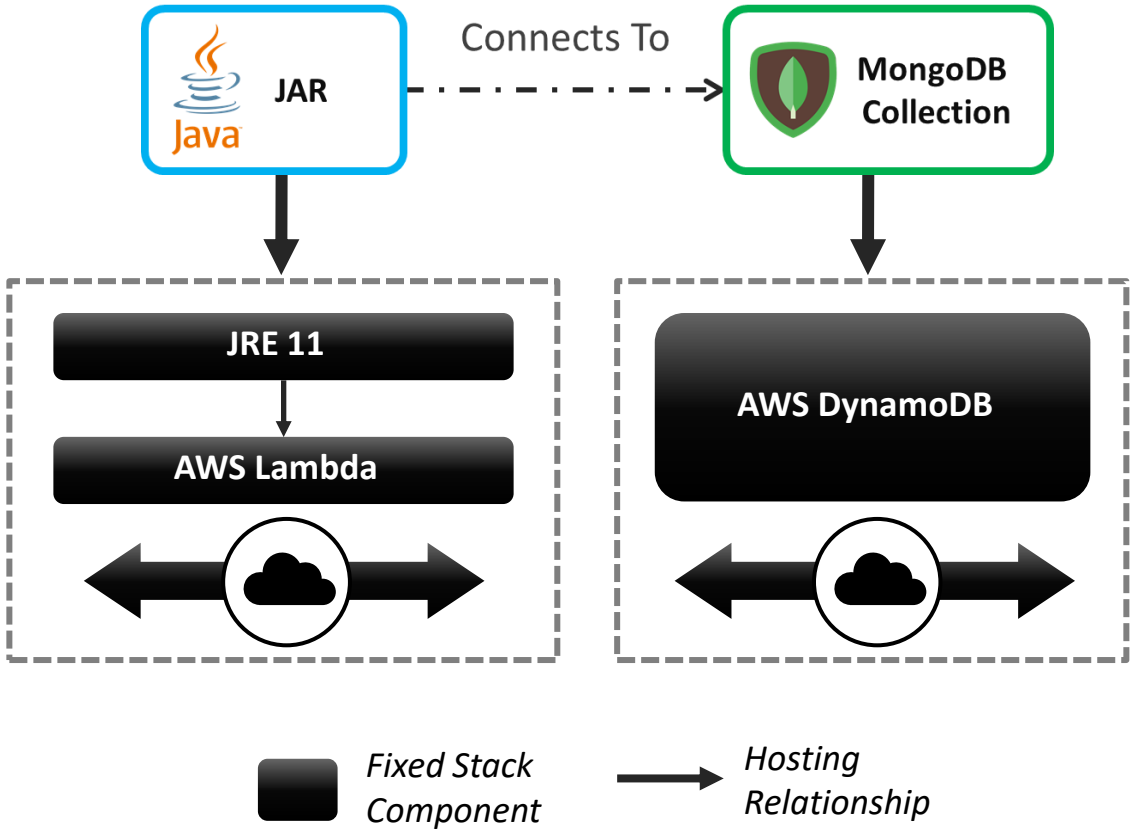
Problem: *How to host a software component that needs to be scaled horizontally but requires no special scaling configuration?*

Context: A software component needs to be hosted on a deployment stack for which cloud consumers do not have special requirements regarding the horizontal scaling behavior, i.e., they want to rely on the provider's default autoscaling mechanism.

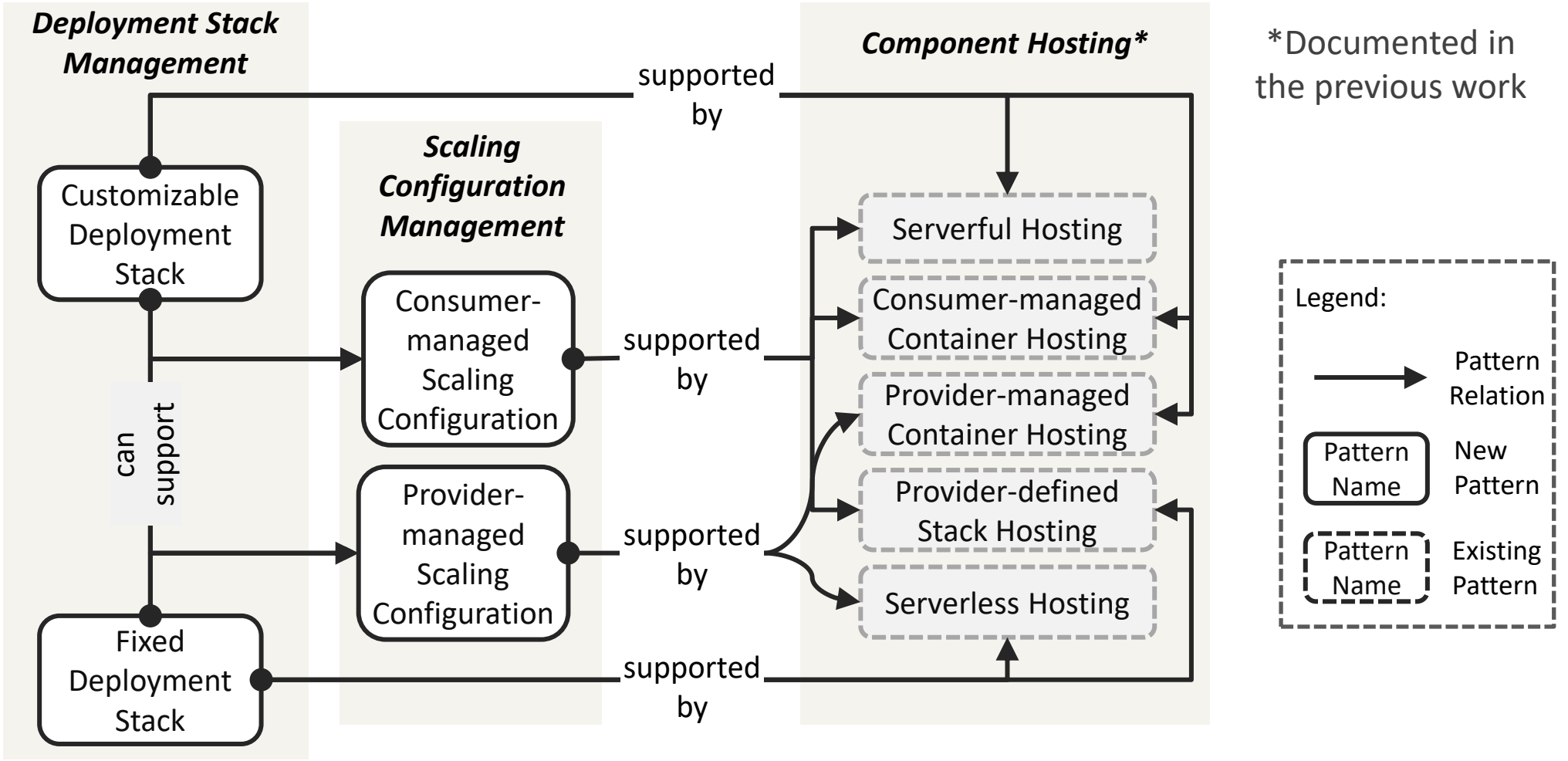
Solution: Host the component on a deployment stack supporting *Provider-Managed Scaling Configuration*, i.e., cloud providers are mainly responsible for the specification of the underlying infrastructure resources and scaling rules.

Known Uses: AWS Lambda, Azure Functions, Google CloudRun, AWS DynamoDB, Azure Blob Storage, IBM Cloud Object Storage

Provider-managed Scaling Configuration Pattern: Example

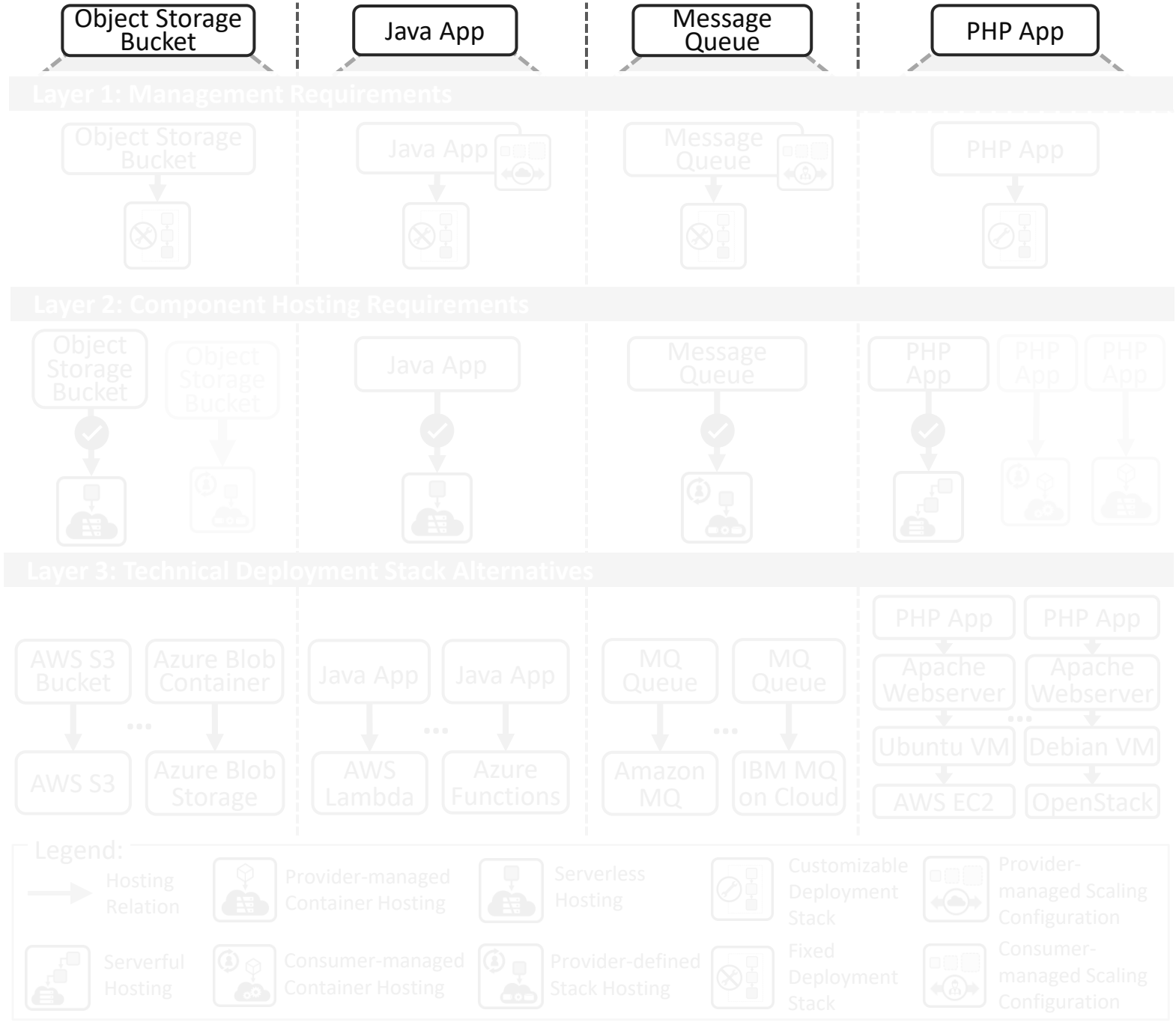


Component Hosting and Management Pattern Language

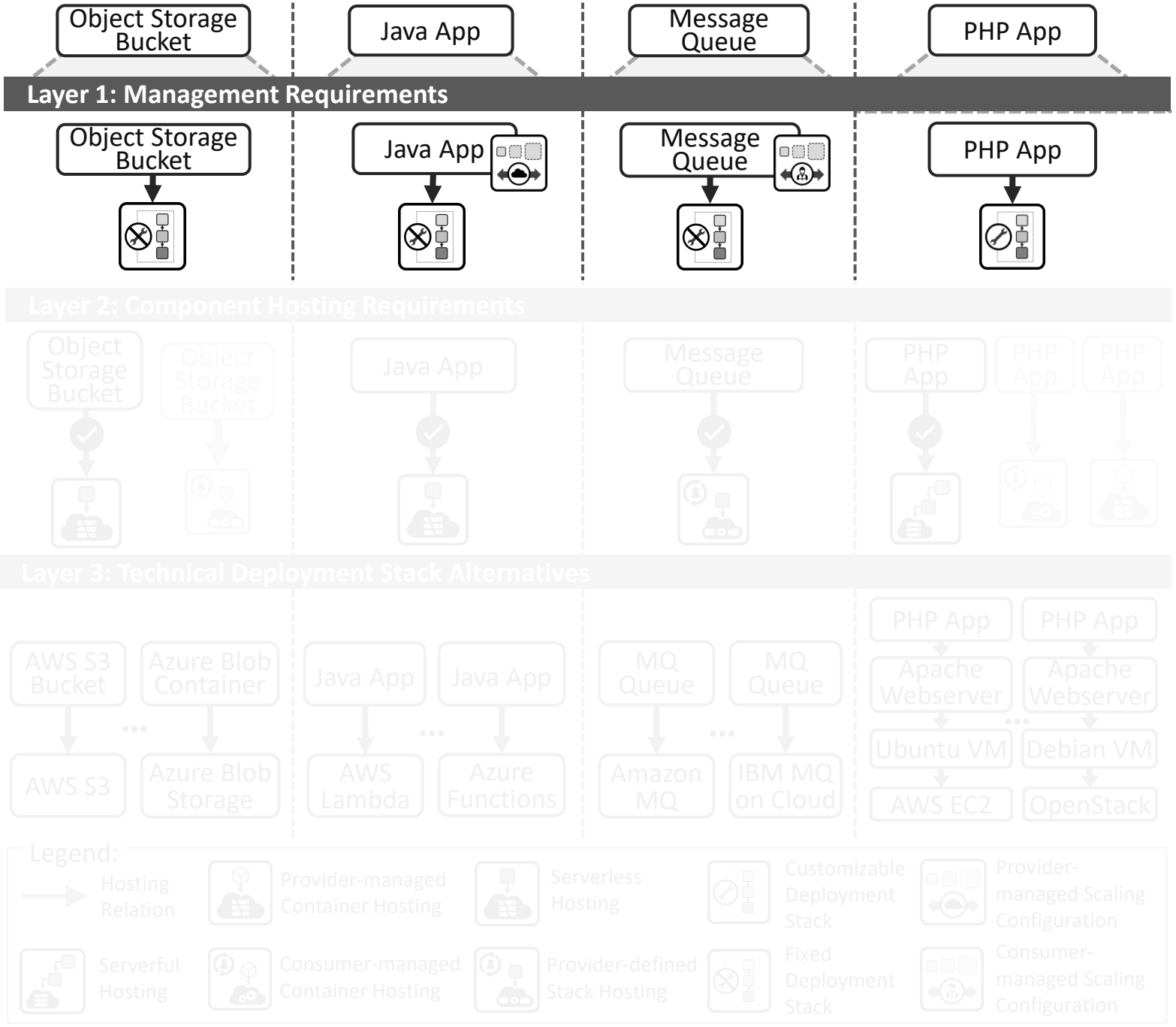


Overview of the relationships among the patterns

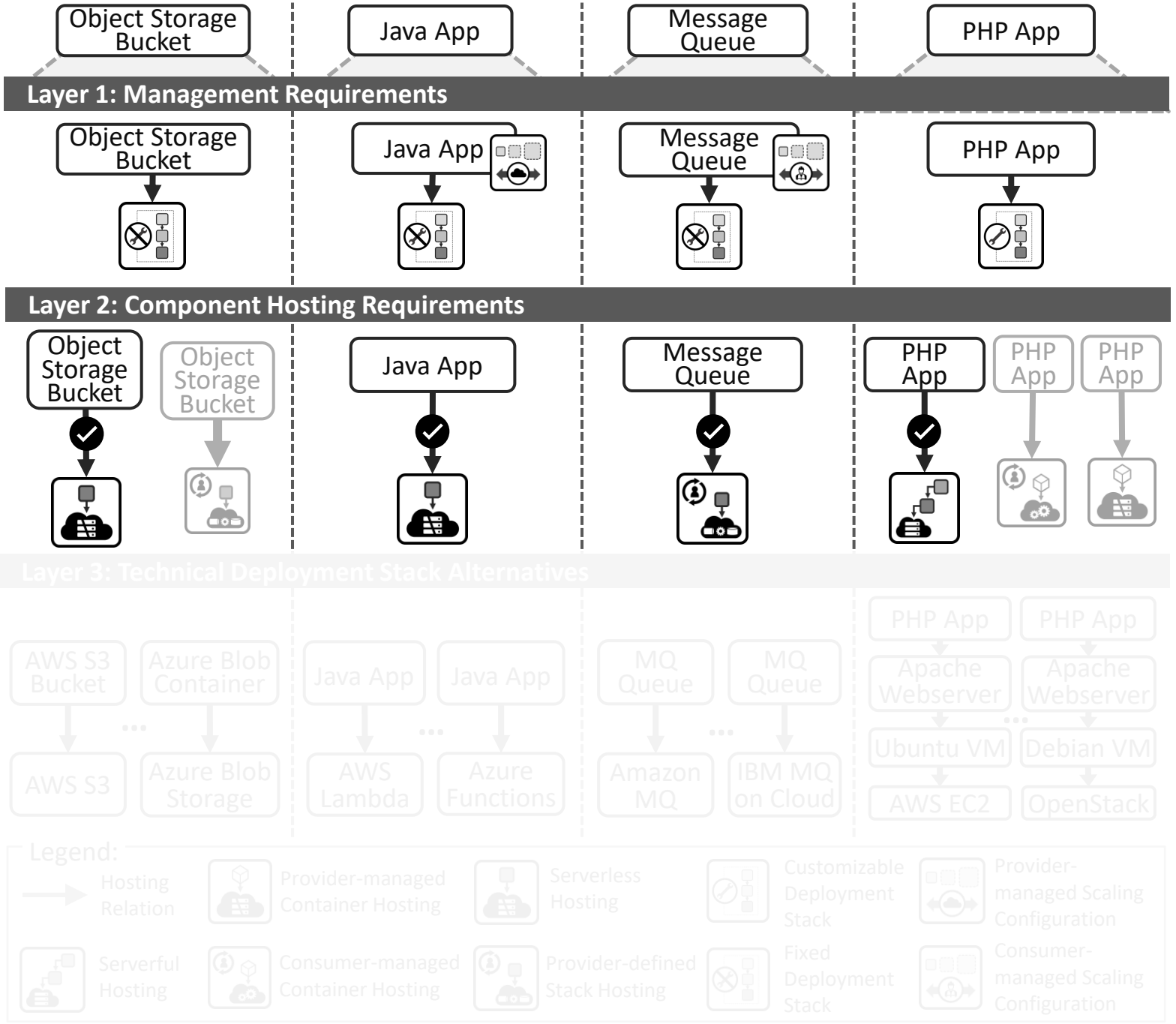
Examples for different component types:



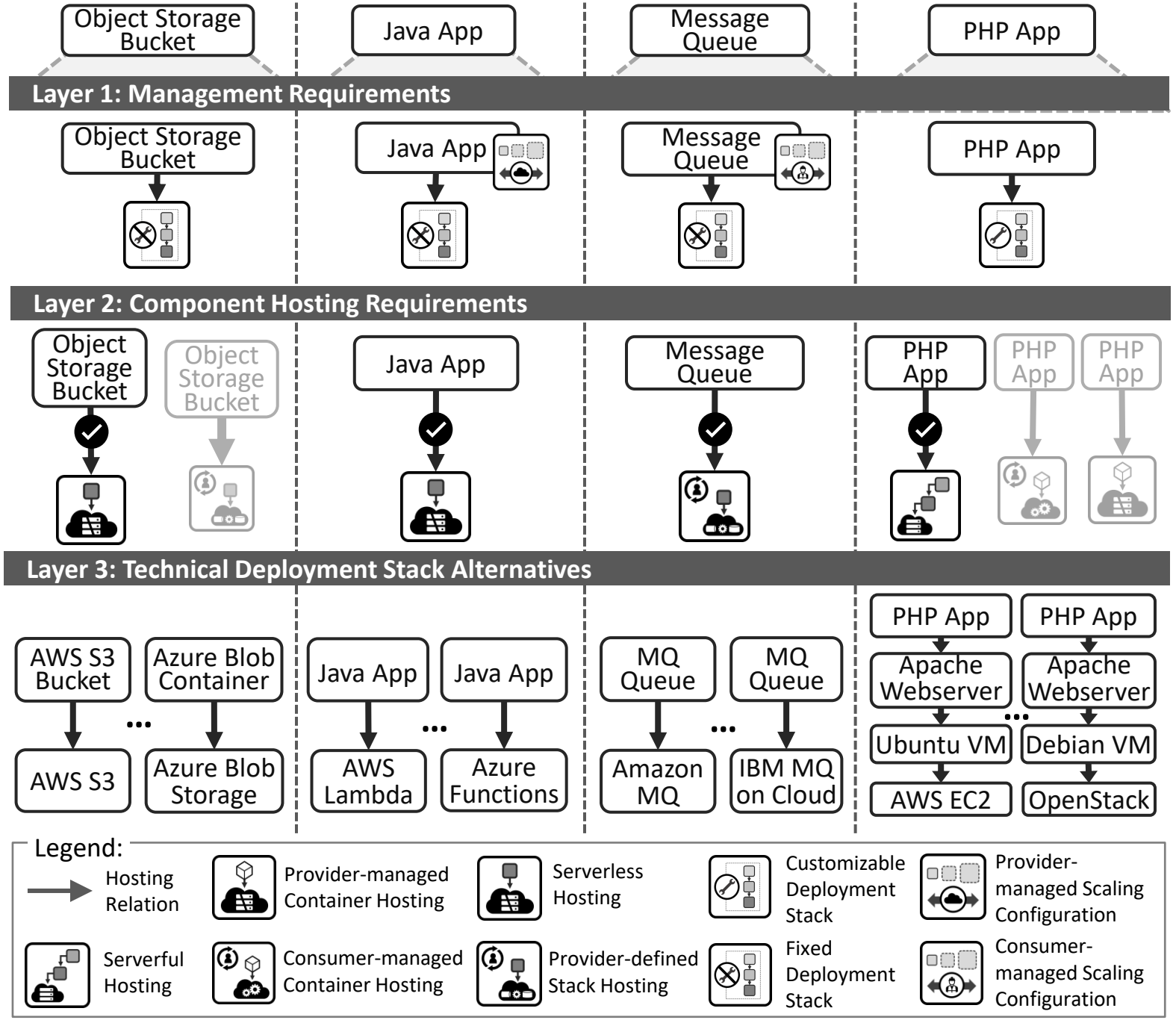
Examples for different component types:



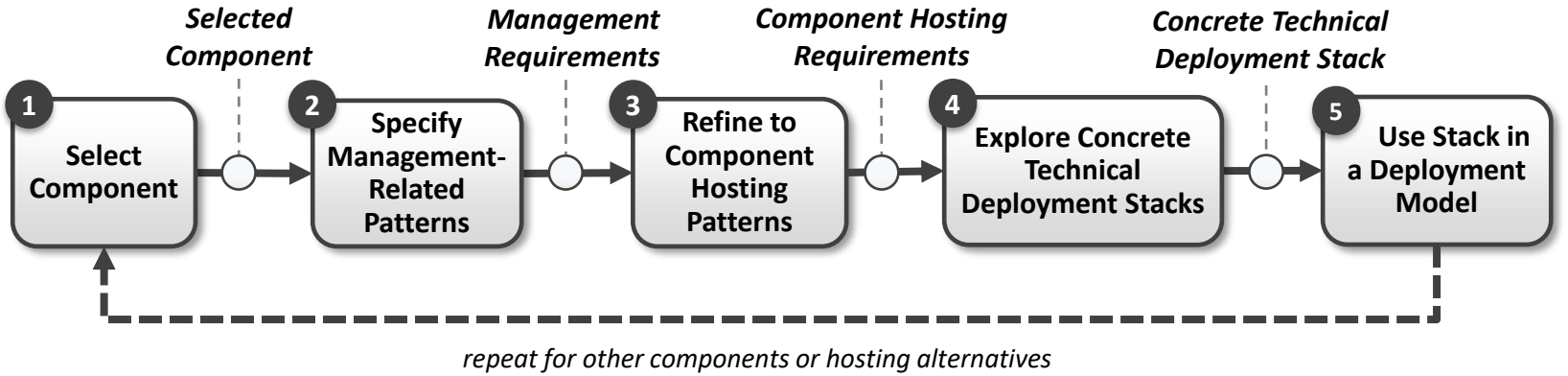
Examples for different component types:



Examples for different component types:



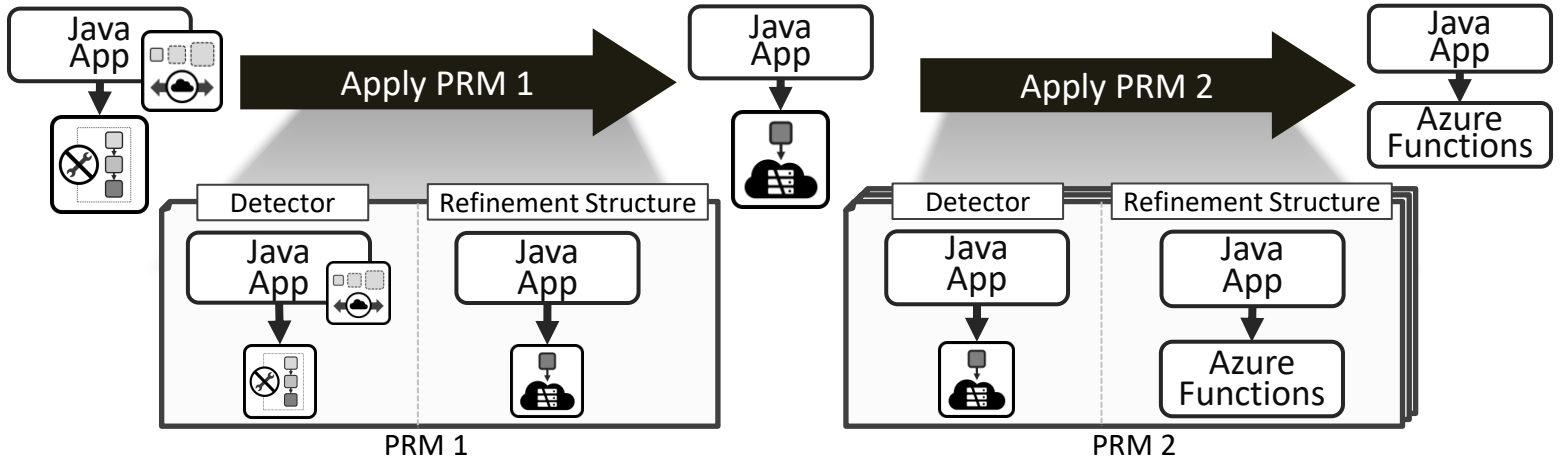
Pattern-based Deployment Stack Exploration



- Associate components with management requirements & refine via pattern relations
  - Express abstract decisions using Deployment Stack Management and Scaling Configuration Management pattern(s), e.g., “JAR” -> “Fixed Deployment Stack”
  - Refine into Component Hosting patterns such as Serverless Hosting, etc.
- Refine into suitable concrete deployment stack variants using automated pattern-refinement approach by Harzenetter et al.*

* Harzenetter, L., et al.: Pattern-based Deployment Models Revisited: Automated Pattern-driven Deployment Configuration. In: Proceedings of the Twelfth International Conference on Pervasive Patterns and Applications (PATTERNS 2020)

Pattern-based Refinement Models



- Two kinds of refinement models
  - To transition between patterns using pattern language relationships
  - To transition from pattern-annotated components to compatible technical deployment stack options contained in the model repository
- Demo using Eclipse Winery: https://youtu.be/-GpGPS5Nc1Q

SCAN ME

* Harzenetter, L., et al.: Pattern-based Deployment Models Revisited: Automated Pattern-driven Deployment Configuration. In: Proceedings of the Twelfth International Conference on Pervasive Patterns and Applications (PATTERNS 2020)

Conclusion & Future Work

- Component hosting and management pattern language capturing abstract and more concrete management trade-offs
- A pattern-based approach for semi-automatic exploration of technical deployment stack alternatives based on abstract management requirements
- Future work
  - “Patterns are never finished”: evolve, refine, capture new patterns for the language
  - Automated exploration of integrated stacks for given application topologies

Thank you!

Vladimir Yussupov
yussupov@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de