



TECHNISCHE
UNIVERSITÄT
MÜNCHEN

Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Declarative Languages

Express the target state
and let the machine figure out the steps.

Markup Languages

Write text intermixed with markup information.

```
foo <em>bar <strong>baz</strong></em> quux
```

Examples:
Wiki, Markdown, AsciiDoc, SGML, HTML, TeX, R[un]off, reStructuredText, RTF



Configuration Languages

Express complex textual configurations.

```
foo bar quux { baz;
quux id 7; baz }
```

Examples:
INI, XML, SXML, JSON, YAML, TOML, HCL



Rule Languages

Express logic and semantic through complex rules.

```
foo(x, y) :- bar(x, y, z)
AND x < 42 AND z >= 10
```

Examples:
SQL, Datalog/RuleML, OWL/SWRL, RIF



Constraint Languages

Find solutions for complex constraints.

```
foo @ bar(X, Y),
baz(X, Y, _) ==> quux.
```

Examples:
MiniZinc, CHR, OCL, Rego, Z3



Query Languages

Retrieve information through paths and expressions.

```
// foo / bar [ @baz ==
"xxx" && @quux > 10 ]
```

Examples:
Glob, RegExp, CSS Selector, XPath, YARA, GraphQL, SQL, SPARQL, Cypher, GQL, ASTq



Validation Languages

Parse and validate complex textual information.

```
foo ::= "bar(#" (?:
[0-9a-fA-F]{2})+ ")
```

Examples:
RegExp, Ducky, BNF, PEG, RELAX NG



solution approach:
execution control:
performance optimization:

automatically, non-obvious
automatically, pre-defined
automatically, pre-defined

Imperative Languages

Express the steps
how the machine has to reach the target state.

Shell Languages

Automate execution of system commands.

```
foo -x 2>&1 | bar -y
--quux <(cat *.cf)
```



Examples:
Korn-Shell, Bourne-Shell, Bash, C-Shell, Batch-Script, PowerShell, AppleScript, DCL



Programming Languages

Execute complex algorithmic steps.

```
for (let i = 0; i < 10;
i++) foo(i, 42)
```



Examples:
JavaScript, TypeScript, Scala, Kotlin, Java, C#, C/C++, Rust, Go, Python, Perl, Ruby, Lua

Text-Processing Languages

Manipulate texts through transformations.



```
/^foo/, /bar.*baz/
$1quux\([0-9]*\)/foo\1/g
```

Examples:
ed, ex, sed, AWK, TXR, XSLT, JSLT



Macro Languages

Pre-process texts with macros.

```
define(`foo', `bar$1baz')
foo(quux)bar
```



Examples:
m4, GPP, CPP, Zoem, ProMac

Expression Languages

Expand path, arithmetic, and boolean expressions.



```
{{ foo.bar[*].baz[42]
.quux + 1 }}
```

Examples:
JQ, YQ, MozJEXL, MathML, JUEL, SpEL



Template Languages

Expand complex text fragments.

```
% for k, v in items %
{{k}}: {{v}}% endfor %
```

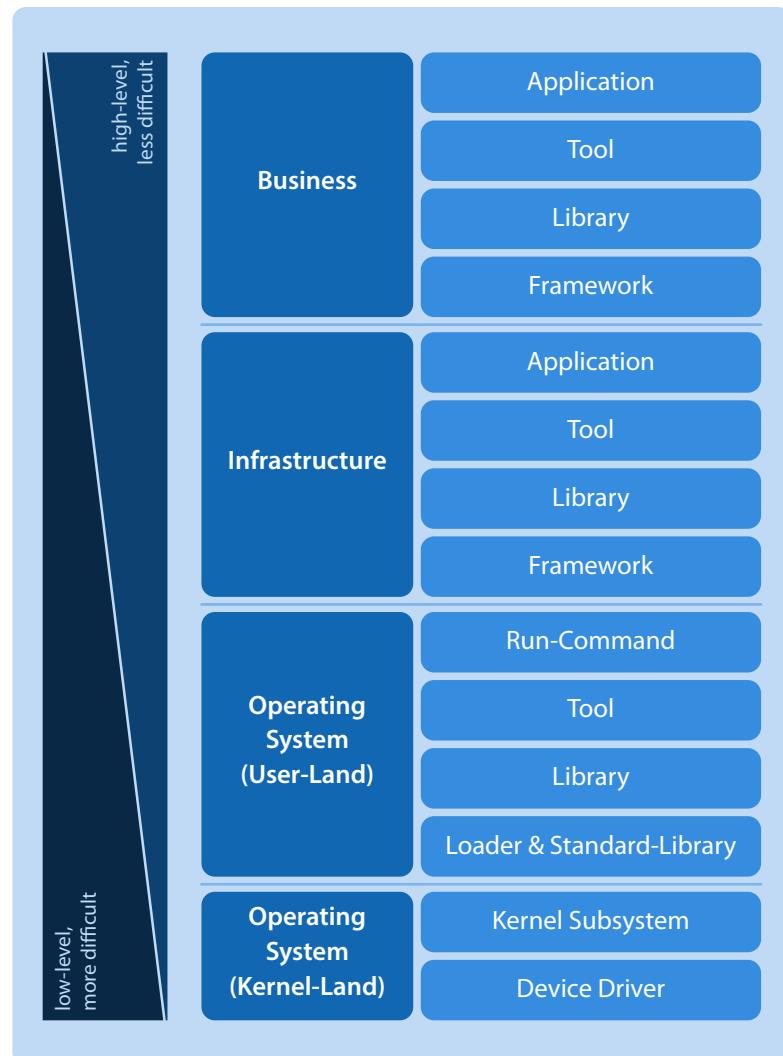


Examples:
Pug, Nunjucks, Handlebars, Mustache, Jinja, Jsonnet

solution approach:
execution control:
performance optimization:

manually, obvious
manually, fine-grained
manually, fine-grained

Technology Platforms



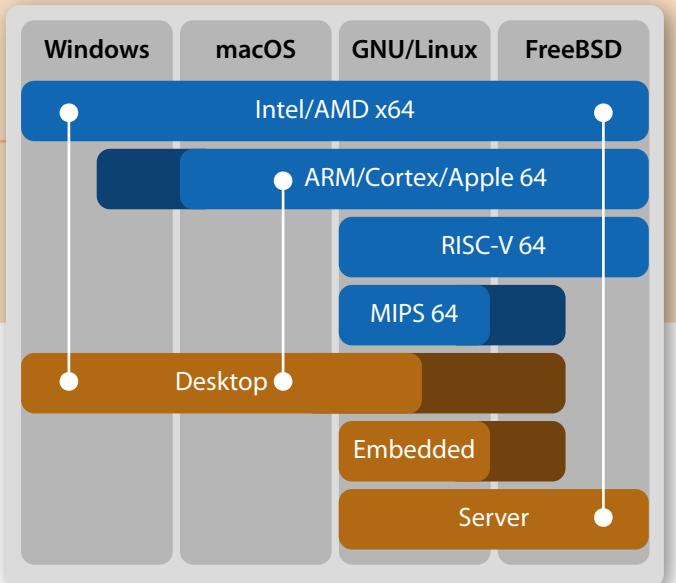
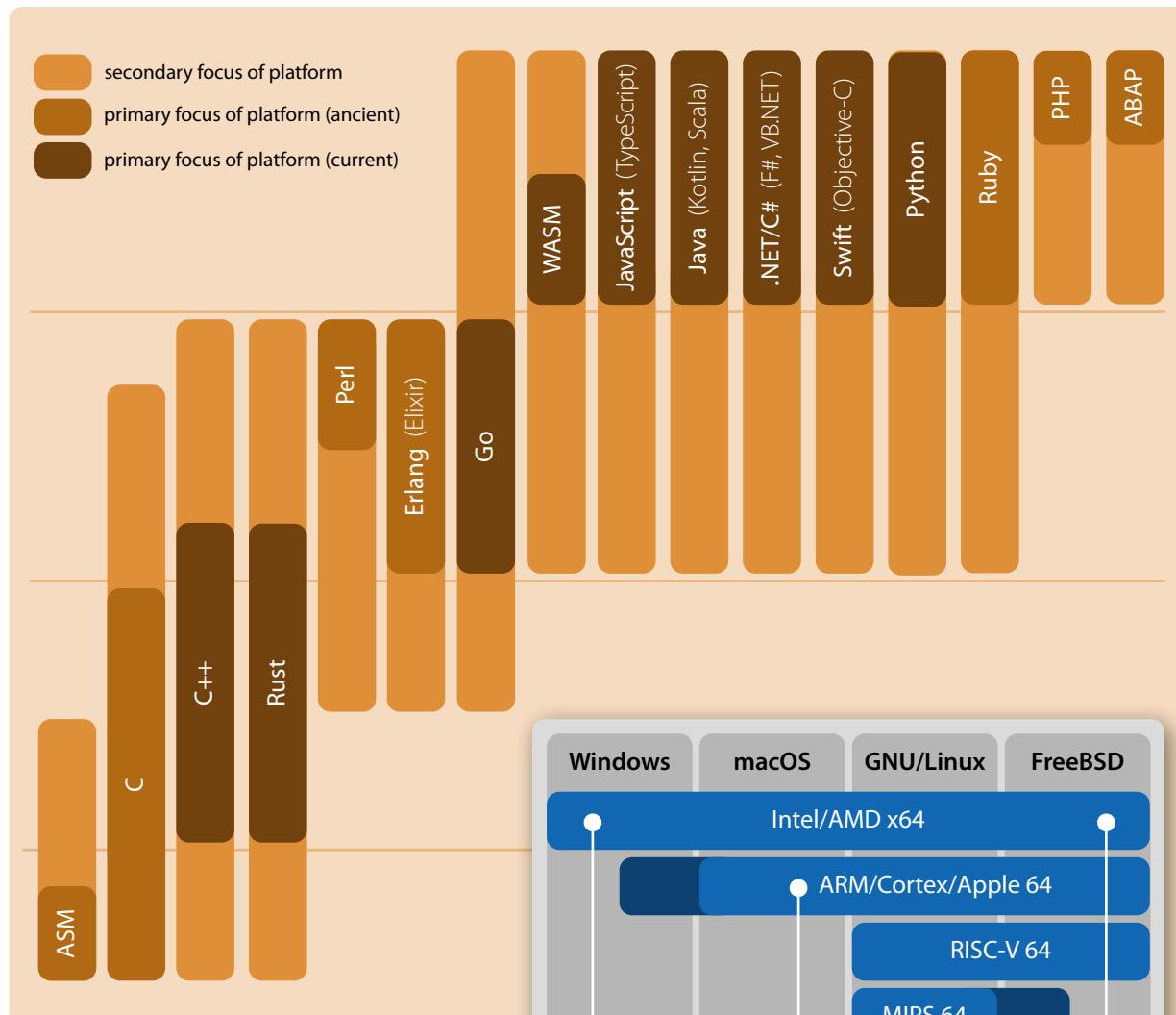
Remember:
A *Technology Platform* is less about choosing a particular programming language and more about choosing a particular ecosystem for targeting a particular level of software!

Opinionated Recommendation (as of 2022):

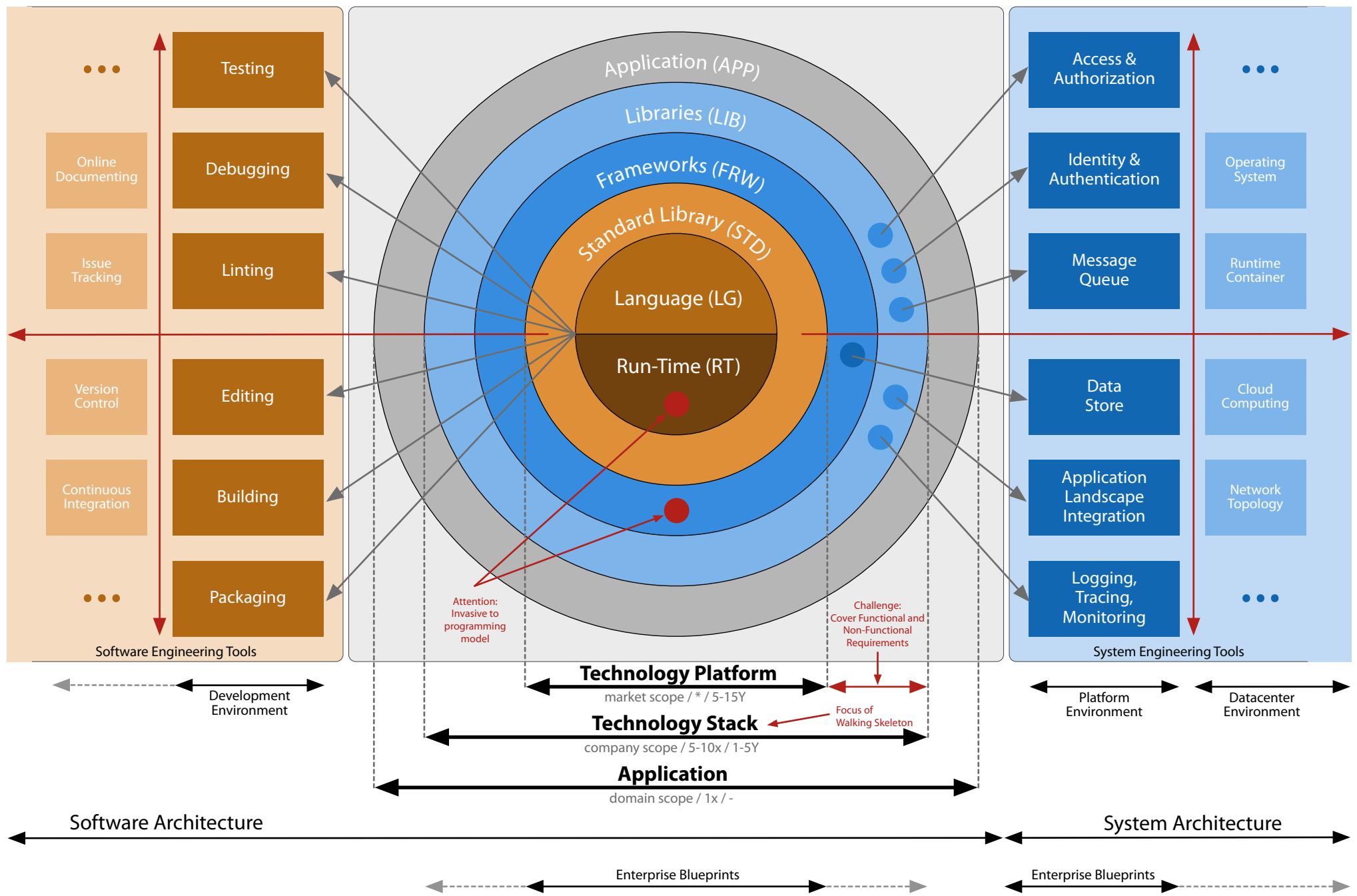
Business:	Scala, Kotlin, TypeScript, AssemblyScript
Infrastructure:	Go, Rust, Scala, Kotlin, TypeScript
Operating System (UL):	Rust, Go
Operating System (KL):	C, C++, Rust

Typical Computing Devices (as of 2022):

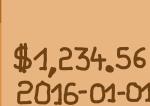
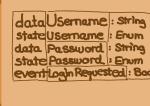
Intel/AMD x64:	Personal Computer (PC)
ARM/Cortex/Apple 64:	Raspberry PI, BeagleBone, ROCKSPro64, iMac
RISC-V 64:	Beagle-V, HiFive Unmatched
MIPS 64:	Compx WPJ344



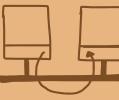
Technology Stack



Rich-Client Aspects

IT Interface Theme			18 Interface Internationalization			DL Dialog Life-Cycle	
Style Reset, Shape, Color, Gradient, Shadow, Font, Icon						Component States, Component State Transitions.	
Bootstrap	TypoPRO, FontAwesome, Normalize					ComponentJS	(none)
IW Interface Widgets			DC Data Conversion			DS Dialog Structure	
Icon, Label, Text Paragraph, Image, Form, Text-Field, Text-Area, Date Picker, Toggle, Radio Button, Checkbox, Select List, Slider, Progress Bar, Hyperlink, Popup Menu, Dropdown Menu, Toolbar, Tooltip, Tab, Pill, Breadcrumb, Pagination, Badge, Alert, Panel, Modal, Table, Scrollbar, Carousel			Value Formatting, Value Parsing, Localization (L10N).			Component, Model/View/Controller Roles, Hierarchical Composition	
Bootstrap	Select2, SlickGrid, ...		VueJS	vue-i18next, I18Next		ComponentJS	ComponentJS-MVC
IL Interface Layouting			DB Data Binding			SP State Persistence	
Responsive Design, Media Query, Frame, Grid, Padding, Border, Margin, Alignment, Force, Magnetism			Reactive, Observer, Unidirectional, Bidirectional, Incremental			Local Storage, Cookies, Caching	
Bootstrap	Swiper, jQuery Page, ...		VueJS	(none)		(none)	Store.js, JS-Cookie
IE Interface Effects			PM Presentation Model			BM Business Model	
Transition, Transformation, Keyframes, Easing Function, Sound Effect, Physics			Parameter Value, Command Value, State Value, Data Value, Event Value, Value Validation, Presentation Logic			Entity, Field, Relationship, Universally Unique Identifiers (UUID)	
VueJS	Animate.css, DynamicJS, Howler, ...		ComponentJS	(none)		(none)	DataModelJS, Pure-UUID
II Interface Interactions			DN Dialog Navigation			UA Use-Case Authorization	
Mouse, Keyboard, Touchscreen, Gesture, Clipboard, Drag & Drop			Deep Linking, Routing, Dialog Flow			User Experience, Dialog Restriction, User, Group, Role, Use-Case, Data, Access.	
VueJS	Hammer, Mousetrap, Dragula, ...		ComponentJS	Director, URL.js		(none)	(none)
IS Interface States			DA Dialog Automation			CN Client Networking	
Rendered, Enabled, Visible, Focused, Warning, Error, Floating			Dialog Macros, Click-Through, Smoke Testing.			Request/Response, Synchronization, Push, Pull, Pulled-Push, REST, GraphQL, Authentication, Session.	
VueJS	(none)		ComponentJS	ComponentJS-Testdrive		(none)	Axios, Apollo Client
IM Interface Mask			DC Dialog Communication			ED Environment Detection	
Markup Loading, Markup Generation, Virtual DOM, Text, Bitmaps, Vectors, 2D/3D Canvas, Accessibility			Service, Event, Model, Socket, Hooks			Runtime Detection, Feature Detection.	
VueJS	jQuery-Markup, D3, Snap.svg, FabricJS, ...		ComponentJS	Latching		(none)	Modernizr, FeatureJS, jQuery-Stage

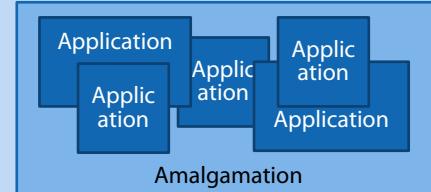
Thin-Server Aspects

ED Environment Detection		SN Server Networking	CN Client Networking
Detect the run-time environment, like underlying operating system, execution platform, network topology, feature toggles, etc.			
Node	process, syspath		
AP Argument Parsing	PI Peer Information	TS Task Scheduling	ET Execution Tracing
Parse options and arguments of the Command-Line Interface (CLI) to bootstrap application parameters.	Determine unique identification and add-on information about the client peer.	Schedule and execute recurring tasks independent of regular I/O operations.	Provide mechanisms for tracing the execution by logging event and measurement information at certain points of interest.
(none)	HAPI	(none)	Microkernel
yargs	hapi-plugin-peer, geoip	node-scheduler	Winston
CP Configuration Parsing	SH Session Handling	DA Database Access	DC Database Connectivity
Load and parse directives from configuration file to bootstrap application parameters.	Manage secured per-connection sessions to keep state between communication requests and/or client sessions.	Map in-memory domain entities onto data store dependent persistent data structure.	Locally or remotely connect the database access layer to the underlying data store.
(none)	HAPI	Sequelize	Sequelize
js-YAML	YAR	GraphQL-Tools-Sequelize	sqlite3, pg
PD Process Daemonizing	UA User Authentication	DS Database Schema	DB Database Bootstrapping
Detach from the startup terminal and host process in order to run fully independently.	Determine and validate the unique identity of the user communicating over the current network connection.	Create, update or downgrade the data schema inside the underlying data store.	Create, update or downgrade both mandatory bootstrapping and optional domain-specific data inside the underlying data store.
(none)	HAPI	Sequelize	Sequelize
daemonize2	JWT, Passport	(none)	ini
PM Process Management	RV Request Validation	RP Request Processing	CC Component Communication
(Pre-)fork child processes and/or threads of execution and monitor and control them during the life-cycle of the application.	Validate the syntactical and semantical compliance of the requests and sanitize the requests.	Process the request by dispatching execution according to the provided request and determined context information.	Provide inter-component communication mechanisms like events, hooks, registry, etc.
(none)	HAPI	GraphQL.js	Microkernel
cluster, nodemon	Joi, DuckyJS		Latching
CM Component Management	RA Role Authorization	RA Role Authorization	CC Component Communication
Structure the code into components, instantiate them under run-time and manage them in a stateful component life-cycle.	Determine whether the role of the current user is allowed to execute the current request.	Determine whether the role of the current user is allowed to execute the current request.	Provide inter-component communication mechanisms like events, hooks, registry, etc.
Microkernel	(none)	GraphQl-Tools-Sequelize	Microkernel
			Latching

AMA Bare Amalgamation

Manually deploy all applications into a single, shared, and unmanaged filesystem location. Dependencies are resolved manually. Examples: Windows Fonts, Unix 1990th /usr/local.

Pro: simple deployment
Con: incompatibilities, hard uninstallation



UHP Unmanaged Heap

Manually deploy all applications into multiple, distinct, and unmanaged filesystem locations. Dependencies are resolved manually. Examples: macOS *.app, OpenPKG LSYNC.

Pro: simple deployment, easy uninstallation
Con: no repair mechanism



MHP Managed Heap

Let individual installers deploy applications into multiple, distinct, and managed filesystem locations. Dependencies are manually resolved or bundled. Examples: macOS *.pkg, Windows MSI, InnoSetup.

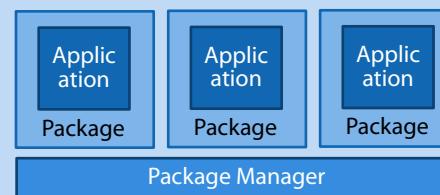
Pro: easy uninstallation, repairable
Con: requires installer, diversity, no dep.



PKG Managed Package

Let a central package manager deploy all applications into a single, shared, and managed filesystem location. Dependencies are automatically resolved. Examples: APT, RPM, FreeBSD pkg, MacPorts, Gradle, NPM.

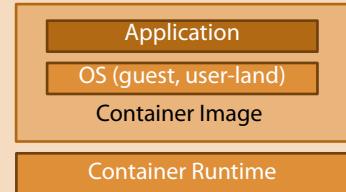
Pro: easy uninstall., repairable, dependencies
Con: P.M. pre-installation, P.M. single instance



CON Container Image

Bundle an application with its stripped-down OS dependencies and run-time environment into a container image. Examples: Docker/ContainerD, Kubernetes/CRI-O, Windows Portable Apps.

Pro: independent, simple deployment
Con: fewer variations, no dependencies

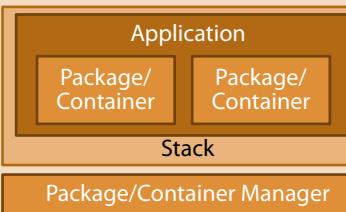


Container Runtime

STK Package/Container Stack

Establish an application out of multiple Managed Packages. Examples: OpenPKG Stack, Docker Compose, Kubernetes/Kompose, Kubernetes/Helm.

Pro: independent, flexible
Con: overhead



Package/Container Manager

VMI Virtual Machine Image

Bundle an application with its full OS dependencies and run-time environment into a virtual machine image and deploy and execute this on a hypervisor. Examples: VirtualBox, VMWare, HyperV, Parallels, QEMU.

Pro: all-in-one, independent
Con: overhead, sealed, inflexible

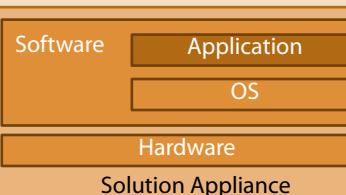


Virtual Machine Hypervisor

APP Solution Appliance

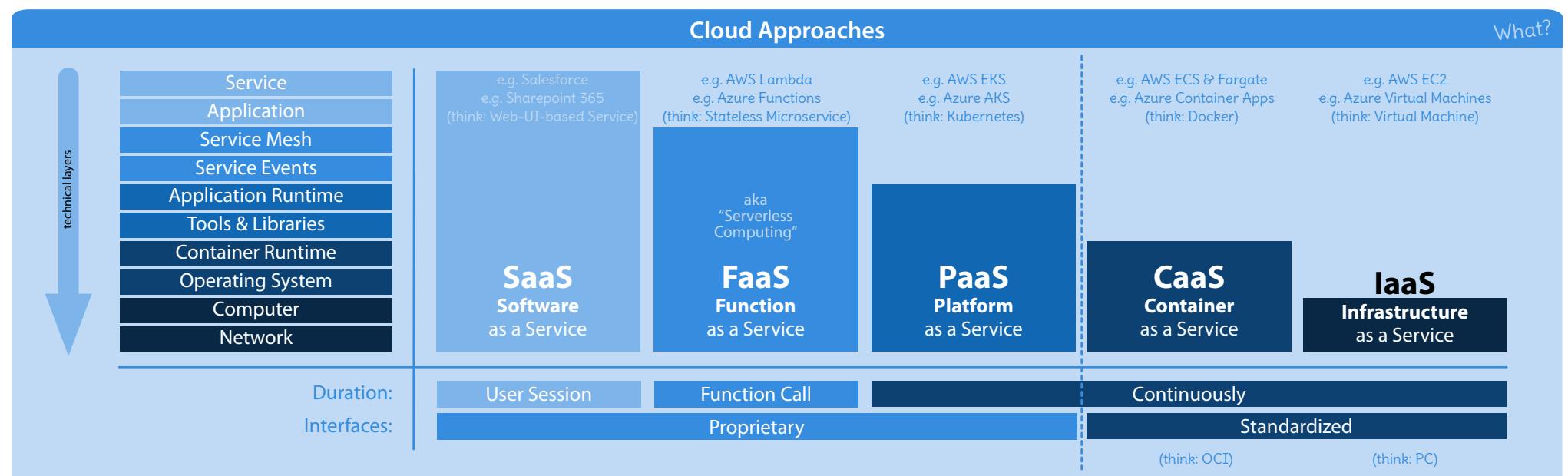
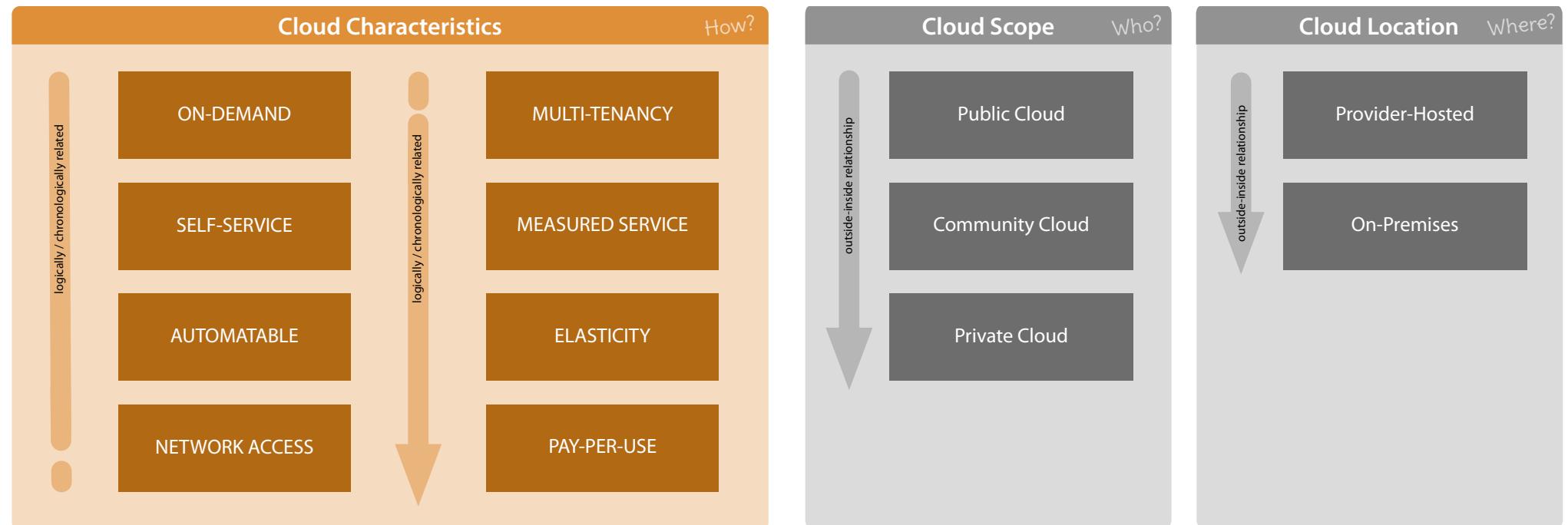
Bundle an application with its full OS dependencies, run-time environment and underlying hardware. Examples: AVM Fritz! Box, SAP HANA.

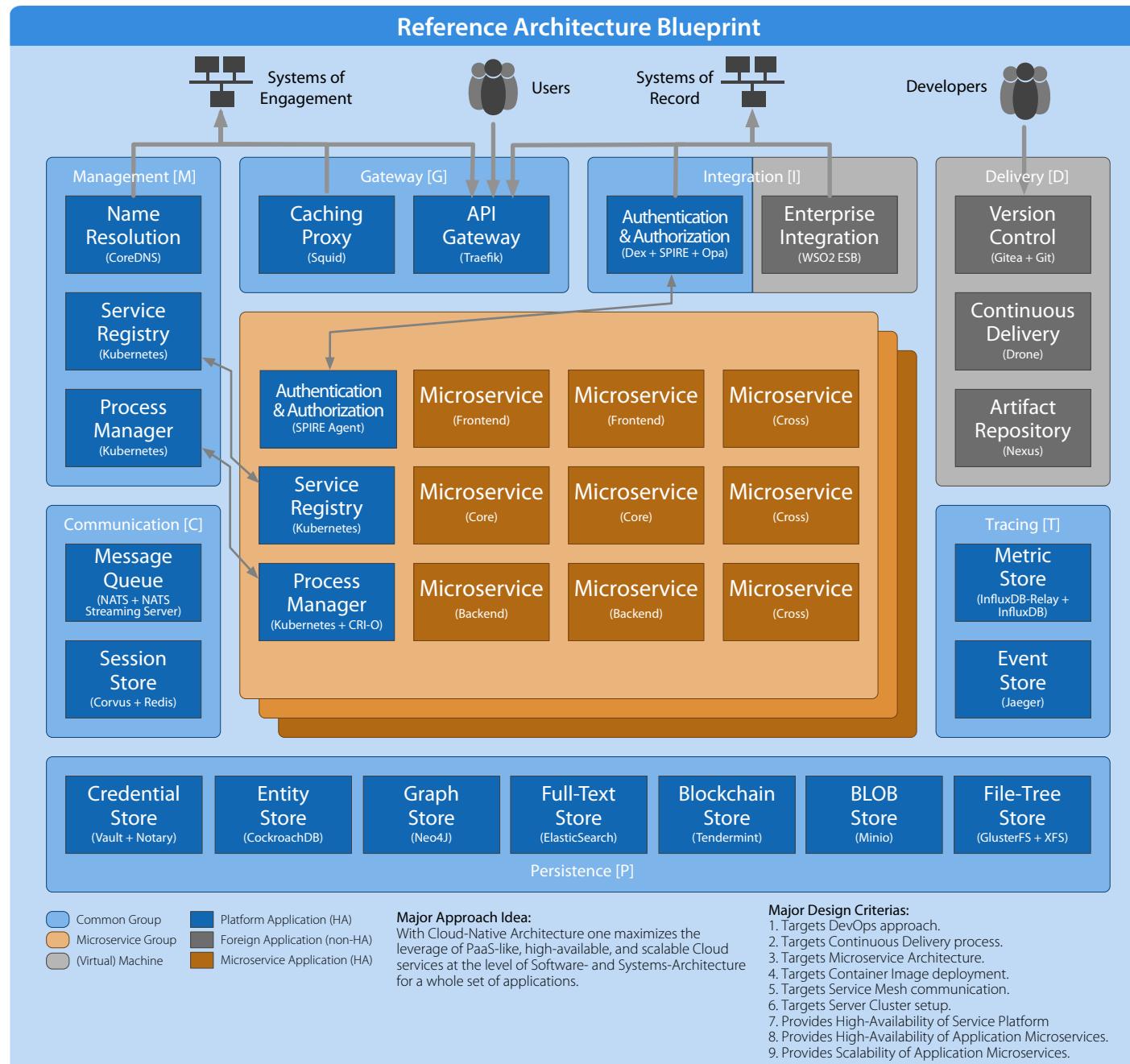
Pro: all-in-one, independent
Con: expensive, sealed, inflexible



Solution Appliance

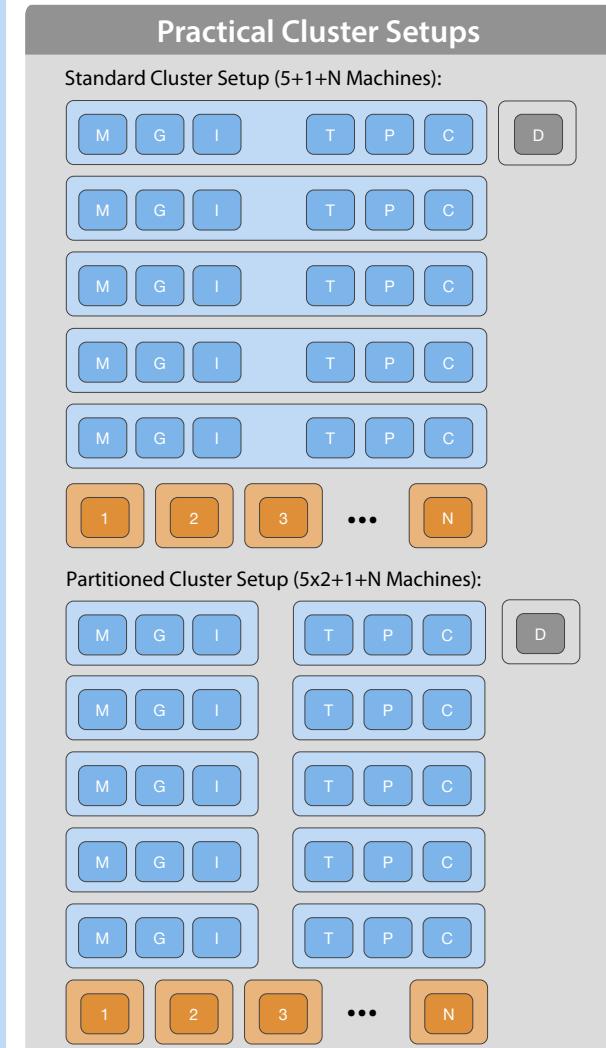
Cloud Computing Resources





CNCF Cloud-Native Definition 1.0

Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal tool.



Offline Capability

