ICS 35.240.01
CCS L 77

NATIONAL STANDARD OF THE PEOPLE'S REPUBLIC OF CHINA

GB/T 44158-2024

Information technology - Cloud computing - Functional requirements of cloud native application support platform

信息技术 云计算 面向云原生的应用支撑平台功能要求

Issued on: June 29, 2024 **Implemented on: January 1, 2025**

Issued by: State Administration for Market Regulation;
Standardization Administration of the PRC

Table of Contents

3 Terms and definitions

The terms defined in GB/T 32400 apply, together with seven terms defined here.

3.1 cloud computing: a paradigm for provisioning and administering a scalable and elastic pool of shareable virtualized resources over a network, on demand and by self-service. A

note lists servers, operating systems, networks, software, applications and storage devices among such resources. The definition is cited from GB/T 32400-2015, 3.2.5.

3.2 cloud native: the set of technologies and methods for designing and building application programs on a cloud computing architecture. A note states that applications built this way are elastic, agile, loosely coupled, easy to deliver and easy to observe.

3.3 artifact: the binary file produced by compiling and packaging source code.

3.4 micro-service application: an application program made up of a group of functionally atomic service components. A note states that such applications run in independent processes, can be deployed independently and communicate through lightweight mechanisms.

3.5 container: multiple independently running kernel units partitioned out of operating system user space. 3.6 container application: an application program built and run on containers.

3.7 function application: an application program built with event-driven computing services. A note states that the developer writes only the business code and does not configure or manage the infrastructure resources the application needs.

5 Overview of the application support platform framework

The cloud native application support platform is described as the technical and software system that supports the digital transformation of an enterprise: agile application development raises development efficiency, fine-grained operation and maintenance raises the degree of automation, and the management of applications, APIs and digital assets carries business innovation.

Following the development and delivery, runtime, operation and maintenance, and management phases of the application life cycle, the functional framework is divided into four parts, shown in Figure 1.

Application development and delivery support capability provides cloud service customers with the technical means to plan, develop and deliver application software efficiently and with agility in a cloud computing environment. Its capability domains are application project management, application software development, application software debugging and testing, application deployment and release, and application orchestration and scheduling.

Application runtime support capability provides the middleware that lets an application system communicate with other systems and cache data. The middleware that is independent of the business mainly covers two capability domains: distributed cache and distributed messaging.

Application operation and maintenance support capability safeguards normal running and

performance tuning, and covers monitoring management, alarm management, performance management and log management.

Application management support capability describes what the platform provides for managing applications from the cloud native application, API and security angles, and what it provides for collaboration between applications on the cloud and between applications on and off the cloud. Its domains are micro-service application management, container application management, function application management, API life cycle management, application security management, and application and data connection management.

6.2 Application software development

6.2.1 Code hosting. Requirements marked as shall: integration of code hosting with the DevOps toolchain; several ways of creating a repository, such as creation from a template or import from SVN; decoupling of repositories, for example one repository per micro-service or separate repositories for common component code and in-house code; protection against code leakage, such as encrypted storage; distributed collaboration, such as version control, branch management and code rollback; the ability to connect to third-party repositories such as GitLab and Gitee. Marked as should: several kinds of code security protection, such as access whitelists and branch protection.

6.2.2 Code development. Marked as shall: a lightweight integrated development environment, with an example of obtaining a cloud development environment through a browser and writing, debugging and running code there; distributed code debugging such as graphical topology, breakpoint inspection and hot code replacement; open interfaces and plug-in extension for integration with third-party systems; traceability of code changes, requirement links and version links. Marked as should: context-aware intelligent code completion.

6.2.3 Code quality analysis. Marked as shall: several kinds of code inspection, such as style, quality and security; inspection of several programming languages, such as C/C++, Java and Python; several kinds of vulnerability inspection, such as IDE inspection, gate inspection and pipeline build inspection; automatic or manual code review with a traceable review process; the ability to integrate code security inspection tools and rules; user-defined inspection rules; quality control such as review checkpoints and user-set thresholds for inspection items. Marked as should: repair suggestions and automatic repair of code defects, and open-source risk analysis and vulnerability detection.

6.2.4 Code compilation and build. Marked as shall: a unified build environment and entry point; fast building, such as distributed C/C++ builds, parallel builds and containerised builds; a build service that developers cannot access or tamper with directly; several languages and software development frameworks; an extensible and traceable build system. Marked as should: user definition of the build environment, covering both platform-configured and self-built environments, and cross-compilation, such as generating

ARM binaries on an x86 platform.

7 Functional requirements for application runtime support capability

7.1 Distributed cache middleware. Marked as shall: distributed caching of pages, session state and application objects; several data storage types, such as key-value pairs and linked lists; high-availability deployment modes such as master-slave and cluster; elastic expansion of cluster and master-standby instances without affecting the upper-layer business during expansion; several ways of use, such as console, CLI or SDK; life cycle management of the data it produces; persistence of final state and operation logs; access control over cached data. Marked as should: orchestration and control of cache instructions, and user authentication with access permission control.

7.2 Distributed message middleware. Marked as shall: basic capabilities such as message backtracking, query, re-delivery and tracing; several message transport protocols, such as MQTT; high-availability deployment such as master-standby and cross-cluster; elastic expansion at cluster node, partition and storage levels; message accumulation and high concurrency; several message types, such as transactional messages and timed messages; message security protection, such as user permission management by message topic and encrypted message transport; different message clearing mechanisms, such as a retention period or a dead-letter queue. Marked as should: life cycle management of the data it produces, statistics on message publication and subscription, and distributed collaboration across clusters and geographic regions.

8 Functional requirements for application operation and maintenance support capability

8.1 Monitoring management shall cover applications, APIs, micro-services, application and service instances and resources; automatic generation and analysis of the application topology; correlation analysis across several monitored objects, a note explaining that monitored objects are the cloud services, middleware and similar items linked to applications, components and environments; classification of monitoring indicators by several dimensions such as resource tag, cluster and image name; user-defined monitoring, such as custom indicators and custom alarm reporting; and one or more ways of presenting results, such as dashboards and topologies. Correlation analysis of different operation and maintenance data for a monitored object, such as indicators, logs and call chains, is marked as should.

8.2 Alarm management shall provide grouping, de-duplication and suppression of alarms; alarm silencing, for example inside a maintenance window; several ways of aggregating alarms, such as collection, direct reporting through an API, or connection of another monitoring system; several notification channels, such as e-mail, SMS and pop-up; several ways of displaying statistical analysis, such as reports, trend charts and share charts; user