E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Sistemas Distribuídos

CÓDIGO: 21108

DOCENTE: Nelson Russo

A preencher pelo estudante

NOME: Edgar Pinto Matuminy

N.º DE ESTUDANTE: 2500691

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 08 de abril de 2026

Questão	Respondida
A1	Não
A2	Sim
A3	Sim
A4	Não
B1	Sim
B2	Não
B3	Não
B4	Sim
C1	Não
C2	Sim
C3	Não
C4	Sim

Nota sobre a experiência profissional do autor

As respostas apresentadas neste E-Fólio incorporam exemplos práticos e configurações reais baseadas na minha experiência profissional como Administrador Middleware.
Atualmente, desempenho funções no Área de Projetos Estratégicos do Governo Vasco (EJIE e Osakidetza), onde administro plataformas críticas 24x7, incluindo servidores de aplicações (Tomcat, JBoss/WildFly, WebLogic, GlassFish, IAS), orquestração de contentores (Docker, Kubernetes, OpenShift), automação de deploys (AnthillPro, CI/CD) e monitorização (Nagios, CheckMK, Pandora FMS, BEM). Anteriormente, trabalhei na GRUPO TECMAN e Quirónsalud, onde iniciei a minha carreira em suporte middleware.

Os conceitos teóricos são fundamentados no livro de Coulouris et al. (2011) e nos Recursos Formativos (RF) da UC. Os exemplos práticos e as métricas apresentadas são ilustrativos do meu dia-a-dia profissional, adaptados para preservar a confidencialidade dos sistemas.

Para mais detalhes sobre o meu perfil profissional, consultar: LinkedIn - Edgar Pinto Matuminy

TRABALHO / RESOLUÇÃO:

A2 - IPv6 e QUIC

Introdução

Na minha experiência como Administrador Middleware (EJIE e Osakidetza), a gestão de redes e protocolos de transporte é uma preocupação diária. Conforme abordado no RF 1.6 (IPv6) e na atualização do RF 2.2 (QUIC) , a transição global para o IPv6 e a evolução do TCP para o QUIC representam mudanças significativas que impactam diretamente o design de sistemas distribuídos modernos. Estes tópicos, que o livro de Coulouris et al. (2011) aborda de forma introdutória, evoluíram significativamente: o IPv6 deixou de ser "solução futura" para ser a realidade da Internet em 2026, e o QUIC (RFC 9000, 2021) tornou-se a base do HTTP/3, substituindo progressivamente o TCP em cenários de alta performance.

Desenvolvimento

(a) Implicações da comunicação direta sem NAT em cenários P2P e IoT

O RF 1.6 destaca que o IPv6 fornece um espaço de endereçamento praticamente ilimitado (2^{128} endereços), eliminando a necessidade de NAT (Network Address Translation). Esta mudança tem implicações profundas:

Aspeto	IPv4 + NAT (atual)	IPv6 (planeado)	Impacto na EJIE (RF 1.6)
Comunicação P2P	Complexa (requer configuração manual de port forwarding)	Direta (cada nó tem IP público)	Facilitaria a sincronização entre clusters de diferentes departamentos
IoT	Gateways NAT introduzem latência e pontos de falha	Dispositivos ligam-se diretamente	Monitorização de sensores em tempo real sem intermediários
Monitorização (Nagios/CheckMK)	Necessita de agentes ou configuração especial para atravessar NAT	Agentes comunicam diretamente com o servidor	Simplificaria a arquitetura de monitorização, conforme referido no RF 1.6

Como o RF 1.6 atualiza, *"a adoção global do IPv6 ultrapassa os 45% do tráfego global, com países como a Índia (70%) e a Bélgica (65%) a liderar"*. Na EJIE, ainda usamos maioritariamente IPv4 com NAT, mas a transição está nos nossos planos, especialmente para cenários de IoT.

(b) Como o QUIC resolve limitações do TCP em microserviços

A atualização do RF 2.2 sobre QUIC explica detalhadamente como o QUIC resolve o *head-of-line blocking*: *"O QUIC é um protocolo de transporte construído sobre UDP que reimplementa, em espaço de utilizador, as garantias de fiabilidade e ordenação do TCP, mas com independência de streams: a perda de um pacote num stream não bloqueia os restantes streams da mesma ligação."* (RF 2.2, atualização)

Na nossa migração para OpenShift, enfrentamos problemas de *head-of-line blocking* em aplicações com múltiplas streams sobre HTTP/2:

Protocolo	Head-of-line blocking	Estabelecimento de ligação (RF 2.2)
TCP + TLS 1.2	Sim	3 RTT
TCP + TLS 1.3	Sim	2 RTT
QUIC (HTTP/3)	Não (independência de streams)	1 RTT (0-RTT para ligações conhecidas)

Como o RF 2.2 atualiza, "o HTTP/3 representa cerca de 30% do tráfego web global em 2025, sendo suportado pelos principais browsers e CDNs". Monitorizamos a performance das nossas aplicações com CheckMK e Pandora FMS para estabelecer uma baseline antes de migrarmos para HTTP/3.

Contextualização

O RF 1.6 refere que *"o World IPv6 Launch (2012) marcou um ponto de viragem, com o tráfego global em IPv6 a crescer mais de 5000% desde então"*. Grandes empresas como Google e Netflix já adotaram QUIC/HTTP/3 em larga escala. Na EJIE, estamos a acompanhar estas evoluções, monitorizando com Nagios e Pandora FMS o impacto potencial antes de adotarmos em produção.

Conclusão

Conforme sintetizado no RF 1.6 e na atualização do RF 2.2, a transição para IPv6 e QUIC representa um passo importante para modernizar a infraestrutura de sistemas distribuídos. O IPv6 elimina as barreiras do NAT, viabilizando arquiteturas P2P e IoT verdadeiramente descentralizadas. O QUIC resolve o *head-of-line blocking* do TCP e reduz a latência de estabelecimento de ligação, sendo particularmente valioso em arquiteturas de microserviços. Na EJIE, monitorizamos estas tecnologias de perto, aguardando maturidade para adoção em produção.

Referências Bibliográficas (A2)

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley. [RF 1.3, RF 2.2]
RF 1.6 – IPv6 e o papel da Internet Society nos Sistemas Distribuídos. (2025/2026). PlataformAbERTA.
RF 2.2 – Comunicação e Suporte em Sistemas Distribuídos – Resumo dos Capítulos 4 a 7. (2025/2026). PlataformAbERTA. [Atualização QUIC]
Internet Society. (2024). *IPv6 deployment: Global trends and analysis*. Internet Society Pulse.
Iyengar, J., & Thomson, M. (2021). *QUIC: A UDP-based multiplexed and secure transport* (RFC 9000). IETF.
Matuminy, E. P. (2026). *Perfil profissional - Administrador Middleware*. LinkedIn. https://www.linkedin.com/in/edgar-pinto-matuminy-215b43278/
***Nota:** Os exemplos de monitorização baseiam-se na minha experiência profissional na EJIE/Osakidetza.*

A3 - Cliente-Servidor vs Peer-to-Peer (P2P)

Introdução

O RF 1.3 (Capítulo 1) e o RF 3.2 (Capítulo 10) estabelecem as bases para comparar os modelos arquiteturais cliente-servidor e peer-to-peer (P2P). Como referem Coulouris et al. (2011), o modelo cliente-servidor tem papéis assimétricos (clientes pedem, servidores respondem), enquanto o P2P se caracteriza pela simetria (todos os nós cooperam como pares). Na minha experiência na EJIE, a escolha entre estes modelos não é binária, mas sim um *continuum* onde adotamos abordagens híbridas consoante os requisitos de segurança, confiabilidade e escalabilidade.

Desenvolvimento

(a) Comparação cliente-servidor vs P2P

O RF 1.3 (secção 1.5) identifica a escalabilidade, o tratamento de falhas e a segurança como desafios fundamentais. A tabela abaixo compara como cada modelo responde a estes desafios:

Critério	Cliente-Servidor	Peer-to-Peer (P2P)	Referência
Segurança	Centralizada (facilita controlo)	Descentralizada (difícil políticas uniformes)	RF 1.3, Cap. 1
Confiabilidade	Dependente do servidor	Elevada (replicação entre pares)	RF 3.2, Cap. 10
Escalabilidade	Limitada pelo servidor	Excelente (cresce com os pares)	RF 3.2, Cap. 10

Na EJIE: A maioria das aplicações segue o modelo cliente-servidor. Por exemplo, o portal de serviços ao cidadão corre em clusters Tomcat e WebLogic com balanceamento Apache/Nginx, monitorizado com Nagios e CheckMK.

(b) Exemplos de P2P moderno
O RF 3.2 (Capítulo 10) apresenta sistemas P2P como o BitTorrent e, na sua atualização, o IPFS (InterPlanetary File System) e a blockchain. Embora o setor público não utilize amplamente P2P, o conceito de replicação de dados inspirou as nossas estratégias de cache distribuída com Redis.
(c) Desafios de segurança distintos
O RF 1.3 (secção 2.4.3) descreve o modelo de segurança, incluindo ameaças a processos e canais:

Modelo	Desafios principais	Mitigação na EJIE
Cliente-Servidor	Ponto único de falha, DDoS	Clusters WebLogic/JBoss, rate limiting
Peer-to-Peer (P2P)	Anonimato, ataques Sybil, falta de auditoria	Inviável no setor público (RGPD)

Como o RF 1.3 salienta, a segurança em sistemas distribuídos exige *"confidencialidade, integridade e disponibilidade"*. No setor público, a centralização é muitas vezes obrigatória para garantir a rastreabilidade e o cumprimento do RGPD.

Contextualização

O RF 1.5 (Linha Temporal) situa o Napster (1999) e o Gnutella (2000) como marcos P2P. Hoje, sistemas como o IPFS (RF 3.2, atualização) representam a evolução deste paradigma. Na EJIE, adotamos uma abordagem híbrida: servidores centrais para autenticação e dados sensíveis, combinados com CDNs e caches distribuídas (Redis) para conteúdos públicos.

Conclusão
Conforme sintetizado no RF 3.2 (Capítulo 10) , os sistemas P2P oferecem *"uma alternativa escalável e resiliente ao modelo cliente-servidor"*. Contudo, no setor público (EJIE, Osakidetza), a centralização é preferível para segurança, auditoria e compliance. A tendência é para arquiteturas híbridas, combinando o melhor de ambos os mundos.

Referências Bibliográficas (A3)
Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley. [RF 1.3, RF 3.2]
RF 1.3 – Fundamentos de Sistemas Distribuídos – Resumo dos Capítulos 1 a 3. (2025/2026). PlataformAbERTA.
RF 1.5 – Fundamentos de Sistemas Distribuídos – Linha Temporal. (2025/2026). PlataformAbERTA.
RF 3.2 – Objetos Distribuídos, Web Services e Sistemas Peer-to-Peer – Resumo dos Capítulos 8 a 10. (2025/2026). PlataformAbERTA. [Atualização IPFS]
Benet, J. (2014). *IPFS - Content Addressed, Versioned, P2P File System*. arXiv preprint arXiv:1407.3561.
Matuminy, E. P. (2026). *Perfil profissional - Administrador Middleware*. LinkedIn. https://www.linkedin.com/in/edgar-pinto-matuminy-215b43278/
***Nota:** Os exemplos de configuração baseiam-se na minha experiência profissional na EJIE/Osakidetza.*

B1 - Protocolos RR vs RRA

Introdução

O RF 2.2 (Capítulo 5) apresenta os protocolos RR (Request-Reply) e RRA (Request-Reply-Acknowledge) como mecanismos fundamentais para invocação remota. Como referem Coulouris et al. (2011), o RR baseia-se em duas mensagens (pedido + resposta), enquanto o RRA adiciona uma mensagem de confirmação (ACK), permitindo ao servidor libertar o histórico de respostas mais cedo. Na minha experiência na EJIE, estes conceitos aplicam-se diretamente na configuração de servidores de aplicações como Tomcat, JBoss/WildFly e WebLogic.

Desenvolvimento

(a) Impacto na gestão de estado no servidor

O RF 2.2 (secção 5.2) explica que no protocolo RR, *"o servidor mantém a resposta no histórico até timeout (longo)"*, enquanto no RRA, *"liberta a resposta após ACK"*. Na prática, configuramos mecanismos equivalentes:

Servidor	Mecanismo	Equivalência com RF 2.2
Tomcat	Sticky session + session replication	RR com timeout configurável
JBoss/WildFly	Distributed web session (Infinispan)	RRA (cache distribuída)
WebLogic	In-memory replication	RRA com replicação

(b) Eficiência em cenários de perda de mensagens

O RF 2.2 descreve as semânticas de invocação: *"maybe (sem garantias), at-least-once (pelo menos uma vez), at-most-once (no máximo uma vez)"*. Monitorizamos perdas com Nagios, CheckMK e Pandora FMS:

Ferramenta	O que monitoriza	Relação com RF 2.2
Nagios	Disponibilidade de portas	Deteção de falhas de omissão
CheckMK	Tempo de resposta HTTP	Deteção de timeouts
Pandora FMS	Logs de erro	Identificação de retransmissões

(c) Semânticas de invocação na prática

O RF 2.2 (secção 5.2) estabelece que *"at-most-once exige histórico de respostas no servidor"*. Na Osakidetza, para operações não idempotentes (ex: registo de consultas), usamos tokens de idempotência (UUID) e cache Redis:

Semântica	Quando usamos	Como garantimos (RF 2.2)
At-most-once	Pagamentos, reservas, registos clínicos	Tokens de idempotência + cache de respostas

Contextualização

O RF 2.2 (Capítulo 5) usa o HTTP como exemplo de protocolo request-reply. Na EJIE, adotámos tokens de idempotência (prática popularizada por Stripe e Netflix) para garantir at-most-once em APIs REST, conforme recomendado na secção 5.2 do RF 2.2.

Conclusão

Conforme sintetizado no RF 2.2 (Capítulo 5) , a escolha entre RR e RRA envolve um compromisso entre memória do servidor e tráfego de rede. Para serviços críticos (Osakidetza), a semântica at-most-once é obrigatória, exigindo cache de respostas (Redis) e monitorização proativa com Nagios/CheckMK.

Referências Bibliográficas (B1)
Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley. [RF 2.2]
RF 2.2 – Comunicação e Suporte em Sistemas Distribuídos – Resumo dos Capítulos 4 a 7. (2025/2026). PlataformAbERTA. [Secção 5.2 – Protocolos request-reply]
Birrell, A. D., & Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1), 39–59.
Matuminy, E. P. (2026). *Perfil profissional - Administrador Middleware*. LinkedIn. https://www.linkedin.com/in/edgar-pinto-matuminy-215b43278/
Nota: *Os exemplos de configuração baseiam-se na minha experiência profissional na EJIE/Osakidetza.*

B4 - TCP/QUIC e XML/JSON/Protobuf

Introdução

O RF 2.2 contém duas atualizações fundamentais para esta questão: a atualização sobre QUIC e HTTP/3 (que substitui progressivamente o TCP na Web) e a atualização sobre serialização (XML → JSON → Protocol Buffers). Como referem Coulouris et al. (2011), a eficiência da comunicação é central em sistemas distribuídos. Na minha experiência na EJIE, estas transições têm impacto direto na latência, largura de banda e escalabilidade das aplicações.

Desenvolvimento

(a) Head-of-line blocking: TCP vs QUIC
A atualização do RF 2.2 sobre QUIC explica:
"O TCP garante entrega ordenada de um fluxo de bytes: se um pacote for perdido, todos os pacotes seguintes ficam bloqueados no buffer do recetor até que o pacote perdido seja retransmitido. O QUIC resolve este problema com independência de streams." (RF 2.2)

Protocolo	Head-of-line blocking	Estabelecimento de ligação (RTT)	Referência
TCP (HTTP/2)	Sim (todas as streams bloqueiam)	3 RTT (TCP + TLS 1.2)	RFC 793; RF 2.2
TCP + TLS 1.3	Sim (todas as streams bloqueiam)	2 RTT	RFC 8446; RF 2.2
QUIC (HTTP/3)	**Não (independência de streams)**	1 RTT (0-RTT para ligações conhecidas)	RFC 9000; RF 2.2 (atualização)

(b) Overhead da negociação TLS
A atualização do RF 2.2 também destaca: *"o QUIC integra nativamente o TLS 1.3, eliminando uma viagem de rede adicional no estabelecimento da ligação segura"*.

Protocolo	RTTs para ligação segura	Referência
TCP + TLS 1.2	3 RTT	RF 2.2
TCP + TLS 1.3	2 RTT	RF 2.2
QUIC (1-RTT)	1 RTT	RF 2.2 (atualização)
QUIC (0-RTT)	0 RTT (para servidores conhecidos)	RF 2.2 (atualização)

(c) Verbosidade do XML vs JSON vs Protobuf
A atualização do RF 2.2 sobre serialização compara os formatos:
"O XML é verboso; o JSON é mais compacto; o Protocol Buffers (protobuf) é binário e tipicamente 3–10× mais compacto que JSON." (RF 2.2)

Formato	Tipo	Tamanho relativo	Uso na EJIE
XML	Textual	Muito grande	Integrações B2B (legado)
JSON	Textual	Médio	APIs REST (portais cidadão)
Protobuf	Binário	Pequeno (3-10× menor)	Microserviços internos (OpenShift)

(d) HTTP/3 + protobuf em microserviços

A combinação HTTP/3 + protobuf é particularmente adequada para microserviços de baixa latência, conforme a atualização do RF 2.2 sugere ao mencionar o gRPC (que usa protobuf + HTTP/2, evoluindo para HTTP/3).

Contextualização
O RF 2.2 menciona que *"o HTTP/3 representa cerca de 30% do tráfego web global em 2025"*. Grandes empresas como Google (gRPC, protobuf, QUIC) e Netflix (HTTP/3) lideram estas transições. Na EJIE, adotámos gRPC/Protobuf para microserviços internos no OpenShift, com ganhos significativos de performance.

Conclusão

Conforme sintetizado nas atualizações do RF 2.2, a evolução do TCP para QUIC e do XML para JSON/Protobuf responde a limitações concretas: head-of-line blocking, overhead de TLS, verbosidade e baixa performance. Na EJIE, adotámos uma estratégia híbrida, monitorizando continuamente com Nagios, CheckMK e Pandora FMS.

Referências Bibliográficas (B4)
Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley. [RF 2.2]
RF 2.2 – Comunicação e Suporte em Sistemas Distribuídos – Resumo dos Capítulos 4 a 7. (2025/2026). PlataformAbERTA. [Atualizações QUIC e Serialização]
Iyengar, J., & Thomson, M. (2021). *QUIC: A UDP-based multiplexed and secure transport* (RFC 9000). IETF.
Google Developers. (2024). *Protocol Buffers: Developer guide*. https://protobuf.dev/
Matuminy, E. P. (2026). *Perfil profissional - Administrador Middleware*. LinkedIn. https://www.linkedin.com/in/edgar-pinto-matuminy-215b43278/
Nota: *Os exemplos de configuração baseiam-se na minha experiência profissional na EJIE/Osakidetza.*

C2 - SOAP vs REST vs gRPC vs GraphQL

Introdução

O RF 3.2 (Capítulo 9) e o RF 3.3 (Recursos Adicionais: Servlet Containers e REST) fornecem a base teórica para comparar SOAP/WSDL, REST, gRPC e GraphQL. Como referem Coulouris et al. (2011), os web services evoluíram de formalismos pesados (SOAP/WSDL) para estilos mais leves (REST) e, mais recentemente, para alternativas de alta performance (gRPC) e flexibilidade (GraphQL). Na minha experiência na EJIE, cada abordagem tem o seu lugar consoante os requisitos de segurança, performance e integração.

Desenvolvimento
(a) Formatos de mensagem e descrição de interface
O RF 3.3 descreve o REST como um estilo arquitetural baseado em recursos e URIs, enquanto o RF 3.2 (Capítulo 9) detalha o SOAP/WSDL:

Abordagem	Formato	Descrição de interface	Referência
SOAP/WSDL	XML (envelope SOAP)	WSDL (formal)	RF 3.2, Cap. 9
REST	JSON	OpenAPI (opcional)	RF 3.3
gRPC	Protobuf (binário)	.proto (IDL)	RF 2.2 (atualização)
GraphQL	JSON (consulta + resposta)	Schema GraphQL	Fontes externas

(b) Segurança: WS-Security vs HTTPS
O RF 3.3 compara os modelos de segurança:
"O SOAP, através das especificações WS-Security, permite aplicar segurança ao nível da mensagem; o REST depende predominantemente de segurança ao nível do transporte (HTTPS)." (RF 3.3)

Aspeto	SOAP (WS-Security)	REST (HTTPS)	Implicação na EJIE
Segurança fim-a-fim	**Sim**	**Não**	Integrações B2B exigem WS-Security
Não repúdio	**Sim**	**Não**	Obrigatório para transações fiscais
Independência de transporte	Sim (HTTP, SMTP, JMS)	Não (depende de HTTP)	Crítico para integrações com sistemas legados

(c) Limitações do REST e surgimento do gRPC e GraphQL
O RF 3.2 (Capítulo 9) , na sua atualização, aborda o declínio do SOAP/WSDL e a ascensão do REST, gRPC e GraphQL:

Limitação do REST	Solução	Referência
Over-fetching / under-fetching	GraphQL	RF 3.2 (atualização)
Ineficiência (JSON textual)	gRPC/Protobuf	RF 2.2 (atualização)
Sem IDL formal	gRPC (.proto)	RF 2.2 (atualização)

(d) Descoberta dinâmica de serviços
O RF 3.2 (Capítulo 9) menciona o UDDI como diretório para web services, mas a sua atualização refere que *"os registos públicos foram encerrados em 2005"*. Hoje, usamos alternativas como o DNS nativo do Kubernetes/OpenShift.

Contextualização
O RF 3.3 conclui que *"SOAP é preferível quando são necessárias garantias formais de contrato, segurança ao nível da mensagem (WS-Security) REST é preferível quando se pretende simplicidade, escalabilidade e interoperabilidade ampla"*. Na EJIE, usamos SOAP/WS-Security para integrações B2B com o Governo Central, REST/JSON para APIs públicas, e gRPC/Protobuf para microserviços internos.

Conclusão

Conforme sintetizado no RF 3.3 e na atualização do RF 3.2, a escolha entre SOAP, REST, gRPC e GraphQL depende dos requisitos. No setor público (EJIE), usamos SOAP para integrações B2B (segurança, não repúdio), REST para APIs públicas (simplicidade) e gRPC para microserviços (eficiência).

Referências Bibliográficas (C2)

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley. [RF 3.2]
RF 3.2 – Objetos Distribuídos, Web Services e Sistemas Peer-to-Peer – Resumo dos Capítulos 8 a 10. (2025/2026). PlataformAbERTA. [Capítulo 9 – Web Services; atualização SOAP/WSDL vs REST]
RF 3.3 – Recursos Adicionais: Servlet Containers e REST. (2025/2026). PlataformAbERTA. [Secção REST; comparação SOAP vs REST]
Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation). University of California, Irvine.
Google Developers. (2024). *gRPC: Introduction*. https://grpc.io/
GraphQL Foundation. (2024). *GraphQL specification*. https://graphql.org/
Matuminy, E. P. (2026). *Perfil profissional - Administrador Middleware*. LinkedIn. https://www.linkedin.com/in/edgar-pinto-matuminy-215b43278/
***Nota:** Os exemplos de integração baseiam-se na minha experiência profissional na EJIE/Osakidetza.*

C4 - Servlet containers e evolução para microserviços

Introdução

O RF 3.3 (Servlet Containers e REST) e o RF 3.4 (Monolítico vs Microsserviços) fornecem a base teórica para esta questão. Como referem Coulouris et al. (2011), os servlet containers exemplificam o princípio da separação de preocupações, delegando no container a gestão da rede, concorrência e ciclo de vida. Na minha experiência como Administrador Middleware na EJIE, administro diariamente Tomcat, JBoss/WildFly e WebLogic, e tenho testemunhado a evolução para microserviços orquestrados por OpenShift (Kubernetes).

Desenvolvimento
(a) Papel do servlet container e ciclo de vida do servlet
O RF 3.3 descreve o ciclo de vida do servlet: *"carrega as classes, instancia os objetos, invoca os métodos de inicialização (*`init()`*), processamento (*`service()`*) e destruição (*`destroy()`*)"*.

Fase	Método	Como gerimos na EJIE (RF 3.3)
Inicialização	`init()`	Verificamos logs no Tomcat/WebLogic
Processamento	`service()`	Monitorizamos latência com Nagios/CheckMK
Destruição	`destroy()`	Verificamos undeploy limpo antes de novo deploy

(b) Gestão de concorrência via thread pooling
O RF 3.3 explica que *"os servlet containers gerem um conjunto de threads pré-criadas (thread pool) que são reutilizadas para tratar pedidos sucessivos"*. Configuro diariamente:

Servidor	Parâmetro	Valor típico	Referência
Tomcat	maxThreads	200-400	RF 3.3
JBoss/WildFly	max-pool-size	50-150	RF 3.3
WebLogic	ThreadPoolSize	25-50	RF 3.3

(c) Separação de preocupações: servlet container vs EJB container

O RF 3.3 estabelece que *"o servlet container abstrai questões de rede, concorrência e ciclo de vida"*, tal como o EJB container abstrai transações, segurança e persistência (RF 3.2, Capítulo 8).

Preocupação	Servlet Container	EJB Container	Referência
Rede	HTTP, sockets	RMI, IIOP	RF 3.3
Concorrência	Thread pool	Pooling de beans	RF 3.3, RF 3.2
Ciclo de vida	`init()`, `service()`, `destroy()`	`@PostConstruct`, `@PreDestroy`	RF 3.3

(d) Evolução para microserviços: Kubernetes e OpenShift

O RF 3.4 (AWS) compara monolítico vs microsserviços, destacando vantagens como *"escalabilidade independente, implantação individual, fraca dependência"*. Na EJIE, temos migrado aplicações monolíticas (EAR/WAR em Tomcat/JBoss) para microserviços em contentores orquestrados por OpenShift:

Responsabilidade	Antes (servlet container)	Agora (OpenShift)	Referência
Gestão de ciclo de vida	Manual (scripts)	Declarativa (YAML)	RF 3.4
Concorrência	Thread pool	Replicação de pods (HPA)	RF 3.4
Descoberta de serviços	Configuração manual	DNS nativo (Services)	RF 3.4

Contextualização

O RF 3.4 afirma que *"os microsserviços ajudam você a inovar com mais rapidez, reduzem riscos, aceleram o tempo de entrada no mercado e diminuem o custo total de propriedade"*. Na EJIE, uma migração real (aplicação de gestão documental) resultou em:

Métrica	Antes (JBoss monolítico)	Depois (OpenShift + microserviços)
Tempo de deploy	15 minutos	30 segundos
Tempo de resposta (pico)	3 segundos	300 milissegundos

Conclusão

Conforme sintetizado no RF 3.3 e no RF 3.4, os princípios de separação de preocupações e gestão de concorrência dos servlet containers foram estendidos ao nível do cluster pelo Kubernetes/OpenShift. Na EJIE, administro ambos os mundos: servidores de aplicações tradicionais (Tomcat, JBoss, WebLogic) e plataformas de orquestração moderna (OpenShift), garantindo a estabilidade 24x7 dos serviços públicos do Governo Vasco.

Referências Bibliográficas (C4)

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley. [RF 3.2, RF 3.3]

RF 3.2 – Objetos Distribuídos, Web Services e Sistemas Peer-to-Peer – Resumo dos Capítulos 8 a 10. (2025/2026). PlataformAbERTA. [Capítulo 8 – Objetos e Componentes Distribuídos]

RF 3.3 – Recursos Adicionais: Servlet Containers e REST. (2025/2026). PlataformAbERTA. [Secção Servlet Containers]

RF 3.4 – Qual é a diferença entre arquitetura monolítica e de microsserviços? (AWS). (2025/2026). PlataformAbERTA.

Apache Software Foundation. (2024). *Tomcat documentation*. https://tomcat.apache.org/

Red Hat. (2024). *OpenShift documentation*. https://docs.openshift.com/

Matuminy, E. P. (2026). *Perfil profissional - Administrador Middleware*. LinkedIn. https://www.linkedin.com/in/edgar-pinto-matuminy-215b43278/

Nota: *Os exemplos de configuração e métricas baseiam-se na minha experiência profissional na EJIE/Osakidetza.*