Beagle Security

Cloud-native application security: the checklist

Application development has moved from monolithic, on premises systems to cloud native architectures built on microservices, containers, orchestration platforms, and managed cloud services. That shift enables faster releases and elastic scaling, but it also multiplies the number of services, APIs, and moving parts that need to stay secure.

Traditional perimeter based security was built for static networks and long lived servers, and it struggles to keep up with systems where infrastructure is defined as code and workloads last minutes instead of years. This checklist is a companion to our guide on cloud native application security best practices, organized around the 4 C's framework: cloud, cluster, container, and code.

Work through each layer with your security, platform, and engineering teams. Tick items when the control is verified in your environment, not when a policy says it should be there.

Cloud - provider configuration and identity

- [ ] **Secure your network and storage configuration**

  Confirm VPC and subnet design keeps traffic properly isolated, firewall rules and security groups are scoped correctly, and storage buckets are locked down rather than open by default. These are still the most common places a cloud environment gets breached.

- [ ] **Enforce encryption and identity controls**

  Confirm encryption is enabled across every service, not just the ones handling obviously sensitive data, and that IAM follows least privilege with MFA, role based access control, and regular access reviews. Check this across every provider in use, since multi cloud setups make it easy for one account to fall behind.

- [ ] **Automate configuration monitoring**

  Confirm a CSPM tool is in place for continuous configuration monitoring, and that policy as code, through OPA or Terraform Sentinel, catches misconfigurations before they reach production. Run automated checks against CIS benchmarks rather than relying on periodic manual review.

Cluster-Kubernetes control plane and workloads

- [ ] **Secure the control plane**

  Confirm the API server requires strong authentication and authorization, RBAC policies are fine grained rather than broad, and admission controllers enforce security policy before workloads run.

- [ ] **Protect etcd and maintain audit trails**

  Confirm etcd is encrypted with access restricted to only what needs it, and that audit logging is comprehensive enough to reconstruct who did what across the cluster.

- [ ] **Apply workload level protections**

  Confirm Pod Security Standards are applied, network policies isolate services from each other, and secrets are managed securely rather than hardcoded. Review service account hygiene, resource quotas, and namespace isolation so a compromised workload cannot move freely.

- [ ] **Adopt cluster benchmarks and policy enforcement**

  Confirm CIS Kubernetes benchmarks are in use, along with a policy enforcement tool such as Kyverno, so configuration drift gets caught automatically rather than during the next scheduled review.

Container-images and runtime

- [ ] **Control what goes into your images**

  Confirm images are built from minimal or distroless bases, scanned before deployment, and signed and verified before they run. Check that no secrets are baked into an image layer.

- [ ] **Secure your registries**

  Confirm registries are secured and access is restricted, so only approved, scanned images can be pulled into production. An unsecured registry undermines every other control upstream of it.

- [ ] **Enforce non root execution and resource limits**

  Confirm privileged containers are disallowed, non root execution is enforced, and resource limits are applied to every workload.

- [ ] **Monitor runtime behavior**

  Confirm runtime behavior is monitored for anomalies and escape attempts, so an issue gets flagged as it happens rather than discovered after the fact.

Code-application and dependency security

- [ ] **Enforce secure coding and strong authentication**

  Confirm secure coding standards are enforced, and that authentication and authorization models are strong enough to hold up under the service to service communication patterns cloud native applications rely on.

- [ ] **Test source code and running applications together**

  Confirm SAST is integrated for source code analysis and DAST is integrated for runtime testing. Static analysis alone cannot catch the misconfigurations or authentication flaws that only surface once the application is actually running.

- [ ] **Test APIs against the OWASP API Top 10**

  Confirm API security testing is mapped to the OWASP API Top 10 and covers REST, GraphQL, and gRPC endpoints, not just the ones with public documentation.

- [ ] **Manage dependencies and supply chain risk**

  Confirm SCA is in place with SBOM generation, and that transitive dependencies are tracked, not just direct ones. Check license compliance at the same time, since it is easy to overlook once a dependency is already in production.

- [ ] **Integrate security testing into CI/CD**

  Confirm security testing runs as part of the pipeline itself rather than as a separate gate before release. This is what makes it possible to catch issues at the pace cloud native teams actually ship.

Pair platform coverage with application layer testing

CNAPP platforms secure your infrastructure and orchestration layers, but they do not replace testing at the application and API layer. Beagle Security runs automated DAST and API security testing built for cloud native environments, covering REST, GraphQL, and gRPC, and integrates directly into CI/CD pipelines so testing keeps pace with how fast cloud native teams ship.

Start a 14 day free trial with Advanced plan features, no credit card required.