ChatGPT

آليات النسخ الاحتياطي وضمان استرجاع البيانات في بيئة SaaS متعددة المستأجرين

مقدمة

في أنظمة **SaaS متعددة المستأجرين** التي تعتمد على MongoDB كقاعدة بيانات، وMinIO لتخزين الملفات، وRabbitMQ كوسيط رسائل، وQdrant كمحرك بحث متجه، يصبح **النسخ الاحتياطي واستعادة البيانات (التعافي من الكوارث)** ضرورة قصوى لضمان عدم فقدان بيانات العملاء. يهدف هذا التقرير إلى استعراض أفضل الاستراتيجيات والآليات لحماية البيانات وضمان إمكانية الاسترجاع الكامل أو الجزئي لأي مستأجر أو مكون من مكونات النظام عند الحاجة. سنغطي استراتيجيات النسخ الاحتياطي العامة، وأنواع النسخ (الكامل والتفاضلي والتزايدي)، وآليات النسخ الاحتياطي لكل مكون (MongoDB, MinIO, Qdrant, RabbitMQ)، بالإضافة إلى الأدوات المفتوحة المصدر المناسبة لبيئة الخوادم الخاصة (VPS)، وإدارة جداول النسخ وتعدد مواقع التخزين، وطرق التشفير، وكيفية استعادة بيانات مستأجر محدد، وأخيرًا أساليب مراقبة عملية النسخ الاحتياطي والتنبيه عند الفشل مع التحقق الدوري من صلاحية النسخ.

أفضل الممارسات العامة للنسخ الاحتياطي والتعافي من الكوارث

لبناء خطة نسخ احتياطي قوية، يجب اتباع مجموعة من **أفضل الممارسات** التي تنطبق على جميع المكونات:

- **النسخ الاحتياطي المنتظم والمجدول:** أخذ نسخ احتياطية بشكل منتظم هو خط الدفاع الأول [1]. يتوجب تحديد جدول زمني يتناسب مع أهمية البيانات وطبيعة تحديثها. على سبيل المثال، إذا كانت البيانات بالغة الأهمية فيجب إجراء نسخ يومي أو حتى بشكل مستمر، أما النظم الأقل أهمية فقد يكفيها أسبوعيًا [2]. من المهم أيضًا جدولة النسخ الاحتياطية في أوقات انخفاض النشاط لتقليل تأثيرها على أداء النظام [2].

- **قاعدة 3-2-1 في النسخ الاحتياطي:** تنص هذه القاعدة على الاحتفاظ بثلاث نسخ من البيانات (إحداها الإنتاجية ونسختان احتياطيتان)، على وسيلتين تخزين مختلفتين، إحداهما خارج الموقع (Off-site) [3] [4]. هذا يضمن عدم وجود نقطة فشل واحدة ويحمي البيانات من الكوارث المحلية (مثل أعطال الأجهزة أو الحريق في مركز البيانات المحلي). في سياق VPS، يمكن تطبيق ذلك عبر الاحتفاظ بنسخة محلية على خادم آخر أو قرص منفصل، ونسخة ثانية على وحدة تخزين سحابية أو مركز بيانات مختلف.

- **تقليل زمن التعطل (Downtime) أثناء النسخ:** بعض أساليب النسخ الاحتياطي التقليدية قد تتطلب إيقاف الخدمة مؤقتًا لضمان الاتساق، لكن هذا غير مقبول عادةً في SaaS. بالتالي يُنصح باستخدام تقنيات تتيح النسخ أثناء عمل النظام (مثل لقطات snapshot لأنظمة الملفات أو استخدام عُقد ثانوية لأخذ النسخ) لتفادي وقف الخدمة [5] [6]. على سبيل المثال، في حالة استخدام MongoDB يمكن تخصيص عُقدة ثانوية مخفية ضمن مجموعة النسخ المتماثل لإجراء النسخ الاحتياطي دون الضغط على العقدة الأساسية [7].

- **اختبارات واستعداد تام:** لا تكتفِ بأخذ النسخ الاحتياطية، بل **اختبر قدرة الاستعادة بشكل دوري**. قم بمحاكاة سيناريوهات الكوارث واستعادة البيانات للتأكد من أن عملية الاسترجاع فعّالة وخطط التعافي محدّثة [8]. الاختبارات المنتظمة تساعد في كشف أي ثغرات في خطة التعافي قبل حدوث كارثة حقيقية. وكما تشير مصادر الخبرة، فإن خطة التعافي لا تكون موثوقة إلا بقدر آخر اختبار ناجح لها [8].

- **الاحتفاظ والإصدارات (Retention & Versioning):** حدد سياسة للاحتفاظ بالنسخ القديمة (مثل الاحتفاظ بنسخ يومية لآخر 7 أيام، وأسبوعية لآخر شهر، وشهرية لآخر سنة). هذا يفيد في حالات استعادة نقطة زمنية معينة أو للتعامل مع أخطاء من نوع حذف/تعديل خاطئ تم اكتشافه بعد أيام. يدعم MinIO مثلاً **ميزة الإصدارات (Versioning)** التي تسمح بالاحتفاظ بنسخ متعددة من الملفات واسترجاع أي نسخة سابقة بسهولة [9] [10] . استخدام الإصدارات المستمرة يعني أن كل تغيير في ملف لا يلغي النسخة السابقة، بل يتم الاحتفاظ بالإصدارات مما يحقق مفهوم **الحماية المستمرة للبيانات (Continuous Data Protection)** [10] [11] . بالطبع يجب إدارة حجم التخزين عبر سياسات حذف للإصدارات القديمة عند انتفاء الحاجة لها.

- **التخزين في مواقع متعددة:** إلى جانب التخزين المحلي، ينبغي تخزين النسخ الاحتياطية في موقع جغرافي مختلف لحماية البيانات من الكوارث الموقعية. يمكن تحقيق ذلك عبر نسخ البيانات إلى خادم VPS آخر في منطقة مختلفة أو استخدام خدمة تخزين سحابية خارجية. على سبيل المثال، يمكن استخدام **النسخ المتماثل المتعدد المواقع** في MinIO لإبقاء نسخة حيّة من البيانات في مركز بيانات آخر بشكل مستمر [12] [13] . سنناقش ذلك بمزيد من التفصيل في قسم MinIO.

- **التأكد من الاتساق (Consistency):** في بيئة متعددة المستأجرين، من المهم ضمان أن النسخة الاحتياطية متسقة تشمل كل المكونات ذات الصلة بنفس النقطة الزمنية. على سبيل المثال، عند أخذ نسخة احتياطية لقاعدة البيانات، يجب التأكد من نسخ الملفات (المخزنة في MinIO) المتصلة بتلك البيانات بنفس فترة النسخ، حتى لا يحدث عدم تطابق بين البيانات الوصفية والمخزون الفعلي للملفات. التنسيق الجيد بين عمليات النسخ الاحتياطي لمكونات النظام (Database, Storage, Queue, Search) ضروري لتحقيق **نسخة نقطة زمنية موحدة** . أحد الحلول هو جدولة النسخ الاحتياطية لكل المكونات في نافذة زمنية محددة ويفضل إيقاف مؤقت لتدفق البيانات (مثل إيقاف مؤقت لمعالجة الرسائل الجديدة) حتى اكتمال أخذ اللقطات.

أنواع النسخ الاحتياطي: الكامل والتفاضلي والتزايدي

عند تصميم استراتيجية النسخ الاحتياطي، لا بد من فهم الفروقات بين الأنواع المختلفة للنسخ:

- **النسخ الكامل (Full Backup):** هو أخذ نسخة شاملة عن كل البيانات. يتم عادةً بصورة دورية (يوميًا أو أسبوعيًا حسب الحاجة). ميزة النسخ الكامل أنه **يتضمن كل شيء** مما يجعل الاستعادة سهلة وسريعة (لا حاجة لتطبيق عدة نسخ متتابعة) [14] [15] . ولكنه في المقابل يستهلك مساحة تخزين كبيرة ويستغرق وقتًا أطول [16] [17] . يعتبر النسخ الكامل أساسًا تعتمد عليه النسخ التفاضلية والتزايدية اللاحقة [18] ، لذلك غالبًا ما يُجرى نسخ كامل بشكل دوري (مثلاً أسبوعيًا) للحفاظ على **نقطة مرجعية** يمكن العودة إليها بسهولة.

- **النسخ التفاضلي (Differential Backup):** في هذا النوع يتم نسخ **جميع التغييرات منذ آخر نسخة كاملة** . أي أنه إذا كان آخر نسخ كامل قد تم في بداية الأسبوع، فالنسخ التفاضلي خلال الأسبوع سيشمل كل التعديلات التي جرت منذ ذلك النسخ الكامل. ميزة التفاضلي أنه أسرع من النسخ الكامل (لأنه لا يكرر البيانات القديمة) وأبسط من التزايدي عند الاستعادة (تحتاج فقط لآخر نسخة كاملة وآخر نسخة تفاضلية لاستعادة النظام). لكن عيبه أن حجم النسخة التفاضلية يزداد بمرور الوقت من النسخة الكاملة؛ كلما طال الوقت منذ آخر نسخ كامل زادت البيانات المتغيرة المضمنة [19] .

- **النسخ التزايدي (Incremental Backup):** ينسخ **فقط البيانات التي تغيرت منذ آخر نسخ احتياطي أُجري (أياً كان نوعه)** [19] . مثلاً إذا كنا نأخذ نسخًا يومية، فكل نسخ تزايدي يومي سيشمل التغييرات منذ النسخ السابق (سواء كان كاملًا أو تزايديًا). ميزة هذا الأسلوب أنه الأسرع والأصغر حجمًا يومًا بيوم، مما يوفر مساحة تخزين وعرض نطاق نقل [20] . لكن الاستعادة تصبح أكثر تعقيدًا وتتطلب سلسلة النسخ (آخر نسخة كاملة وكل النسخ التزايدية التالية لها حتى النقطة المرغوبة). لتحقيق استعادة سليمة يجب تطبيق النسخ الكامل ثم تطبيق كل **النسخ التراكمية** التي تليه بالترتيب الزمني.

كثيرًا ما تستخدم **مزيجًا من هذه الأنواع** في الخطة: مثلًا **نسخ كامل أسبوعي مع نسخ تزايدي يومي** [21]. بهذه الطريقة يكون حجم النسخ اليومي صغير وزمنه قصير، في حين تؤمن النسخة الأسبوعية الكاملة مرجعًا للتفاضل أو للتزايد في الأسبوع التالي. يمكن أيضًا استخدام النسخ التفاضلي مع الكامل (مثلاً كامل شهري وتفاضلي يومي)، إلا أن التزايدي أكثر شيوعًا إذا كانت أدوات النسخ تدعم تتبع التغييرات بدقة (كاستخدام سجل المعاملات أو الـOplog في حالة MongoDB). اختيار الطريقة المثلى يعتمد على حجم البيانات ومعدل التغيير ومتطلبات **نقطة الاستعادة المستهدفة (RPO)** و **زمن الاستعادة (RTO)**. فمثلًا النسخ التزايدي يوفر نقاط استعادة أقرب (RPO صغير) لكنه يزيد زمن الاستعادة (RTO) بسبب وجوب تطبيق سلسلة النسخ، بينما النسخ الكامل يعطي أسرع RTO لكنه قد يفوت بعض التغييرات لو كان الجدول متباعدًا. لذا الموازنة مطلوبة. جدير بالذكر أن MongoDB Cloud Manager وOps Manager يتيحان **نسخًا مستمرًا باستخدام الـOplog** لتحقيق نقطة زمنية دقيقة، وهي مكملة للنسخ الكامل الدوري [21].

النسخ الاحتياطي لقواعد بيانات MongoDB

MongoDB هي المخزن الرئيسي للبيانات، وفقدانها يمثل الخطر الأكبر. في بيئة متعددة المستأجرين، قد تكون البيانات إما في قاعدة بيانات واحدة مع تمييز كل مستأجر (مثلاً بحقل tenant_id) أو موزعة على قواعد بيانات/مجموعات منفصلة لكل مستأجر. في كلتا الحالتين، استراتيجيات النسخ الاحتياطي تشمل ما يلي:

- **استخدام أدوات MongoDB الرسمية:** أداة `mongodump` المدمجة تتيح أخذ نسخة احتياطية من قاعدة البيانات وحفظها (بصيغة BSON ضغوط غالبًا). هذا الخيار مناسب **للمجموعات الأصغر أو المتوسطة** الحجم، وهو بسيط الاستخدام. ومع أنه لا يدعم النسخ التزايدي مباشرة، إلا أنه يمكن استخدام خيار --oplog لحفظ سجل العمليات بالتوازي مع النسخ الكامل لمتابعة التغييرات لاحقًا [22]. عيب `mongodump` أنه قد يستهلك وقتًا وقد يؤدي إلى حجم ملفات كبير على قواعد البيانات الضخمة، إضافة لكونه **نسخًا كاملاً فقط** (لا يدعم التفاضلي/التزايدي بشكل مباشر) [23] [24]. في حالة البيانات الكبيرة جدًا، قد يصبح أخذ dump يومي غير عملي بسبب الوقت والحجم.

- **لقطات نظام الملفات (Filesystem Snapshots):** إذا كان MongoDB يستخدم محرك التخزين WiredTiger (الافتراضي) فيمكن الاستفادة من **لقطات تخزين** على مستوى نظام الملفات أو الكتلة (مثل LVM snapshot أو لقطة الآلة الافتراضية). هذه الطريقة سريعة جدًا وتأخذ **صورة نقطة زمنية** للبيانات دون الحاجة لتصدير كل الوثائق [25]. يجب قبل أخذ اللقطة التأكد من اتساق البيانات إما عبر وضع MongoDB في حالة `fsyncLock()` (لقفل الكتابة مؤقتًا) [26] أو باستخدام عقدة ثانوية في وضع القراءة فقط. بعد أخذ اللقطة يمكن فك القفل. هذه الطريقة ممتازة لقواعد البيانات الكبيرة حيث النسخ قد يستغرق وقتًا طويلًا. كما أنها تدعم **النسخ التزايدي** إذا كان نظام التخزين يدعم ذلك (مثل ZFS incremental snapshots أو استخدام أداة نسخ ملفات تفاضلية). تشير وثائق MongoDB إلى أن **النسخ عبر اللقطات** يضمن اتساقًا كاملاً ويوفر إمكانية النسخ التزايدي اعتمادًا على تقنية التخزين المستخدمة [21].

- **النسخ المتماثل Replication واستخدام عقد مخصصة للنسخ:** من **أفضل الممارسات** أن يتم نشر MongoDB كعنقود **متماثل (Replica Set)** حتى في بيئة VPS. وجود عقدة ثانوية (غير مرئية للتطبيقات Hidden Secondary) يسمح بأخذ النسخ الاحتياطية منها دون التأثير على العقدة الأساسية [7]. يتم إعداد عقدة مخفية بحيث لا تشارك في انتخابات القيادة ولا تتلقى استعلامات قراءة من التطبيق، ثم يمكن تشغيل `mongodump` أو أي أداة نسخ عليها بحرية. هذا يزيل عبء النسخ من قاعدة البيانات المنتجة. علاوة على ذلك، النسخ المتماثل بحد ذاته هو شكل من أشكال حماية البيانات: إذ أن فقدان عقدة لا يعني فقدان البيانات طالما توجد عقدة أخرى تحتوي على نسخة منها. ولكن النسخ المتماثل ليس بديلًا عن النسخ الاحتياطي لأنه إذا حدث حذف بيانات عن طريق الخطأ أو تلف منطقي، فسيتكرر على جميع العقد. لذا لا بد من وجود نسخ خارجية تاريخية.

- **أدوات متقدمة مفتوحة المصدر:** يوجد أدوات مفتوحة المصدر متخصصة في النسخ الاحتياطي لـMongoDB. مثلًا **Percona Backup for MongoDB (PBM)** تتيح نسخًا احتياطيًا مجدولًا يتضمن **نسخًا كاملة وتراكمية باستخدام سجل العمليات (oplog)**، مما يحقق نسخًا تزايديًا فعالًا. كما تدعم PBM النسخ إلى مساحات تخزين متعددة (مثل ملفات محلية أو S3 متوافق مثل MinIO). استخدام أدوات مثل

PBM أو `mongobackup` يمكن أن يبسط إدارة النسخ خاصة في بيئة متعددة العقد. أيضًا أداة **MongoDump with Point-in-Time Recovery** عبر الـOplog هي أسلوب شائع (أي أخذ dump كامل دوري مع تسجيل الـOplog بشكل مستمر لإمكان استعادة نقطة محددة بين النسخ الكاملة) [19] [27] .

- **اعتبارات خاصة متعددة المستأجرين:** إذا كان لكل مستأجر **قاعدة بيانات مستقلة** في MongoDB، يمكن تنفيذ نسخ احتياطية مجزأة لكل قاعدة (مثلاً إجراء `mongodump` على قاعدة مستأجر معين حسب الحاجة أو وفق جدول خاص إذا اختلفت أهميتهم). هذا العزل يُسهل أيضًا **استعادة مستأجر واحد فقط** دون التأثير على الآخرين – سنغطي ذلك لاحقًا. أما إذا كان المستأجرون يشتركون في قاعدة واحدة مع تمييز داخلي، فيجب الاعتماد على النسخ الكاملة ثم استخراج بيانات المستأجر عند الاستعادة (بإجراء استعلامات لاسترجاع وثائق tenant محدد أو استخدام **مرشحات أسماء الفضاء** التي يوفرها MongoDB عند النسخ [28]). على سبيل المثال، تدعم أدوات MongoDB تصفية النسخ الاحتياطي على مستوى مجموعة collection أو قاعدة محددة [28] ، وهذا يمكن استعماله لعزل بيانات مستأجر معين أثناء عملية الاسترجاع.

- **التشفير والأمان:** تأكد من **تشفير النسخ الاحتياطية لقاعدة البيانات** وهي في حالة الراحة (at rest). MongoDB يدعم تشفير البيانات على مستوى القرص، ولكن لزيادة الأمان يمكن أيضًا **تشفير ملفات النسخ الاحتياطي** نفسها (مثلاً تشفير أرشيف `mongodump` بكلمة سر باستخدام GPG أو أدوات النسخ التي تدعم التشفير). هذا مهم خصوصًا عند خزن النسخ في مواقع خارجية أو سحابية لضمان عدم تمكن أي جهة من قراءة البيانات الحساسة [29] [30] . سنناقش التشفير بمزيد من التفصيل لاحقًا.

باختصار، استراتيجية MongoDB قد تكون: **نسخ كامل أسبوعي** (عبر mongodump أو لقطة)، **نسخ تزايدي يومي** (عبر Oplog أو أدوات PBM)، تخزين النسخ في خادم محلي وخارجي، واختبار استعادة دوري. يجب أيضًا مراقبة أداء قاعدة البيانات أثناء النسخ لتجنب التأثير على المستخدمين. استخدام عتاد قوي ووسائط سريعة للنسخ (مثل أقراص NVMe) أو جدولة النسخ في أوقات الخمول يساعد في ذلك. كما يُنصح بتفعيل الضغط (Compression) في النسخ لجعل حجم الملفات أصغر ونقلها أسرع [31] .

النسخ الاحتياطي لمخزن الملفات (MinIO Object Storage)

يُعد **MinIO** بديلاً ذاتي الاستضافة لخدمات التخزين الكائني (مثل S3)، ويستخدم لتخزين ملفات المستخدمين ومرفقاتهم. فقدان البيانات في MinIO قد يعني ضياع مستندات أو صور العملاء، مما لا يقل خطورة عن فقدان قاعدة البيانات. استراتيجية حماية البيانات في MinIO ترتكز على **التكرارية (Replication)** و**النسخ الخارجي**، وكذلك ميزات مضمنة مثل **الإصدارات (Versioning)**. فيما يلي أبرز الآليات:

صورة توضح مفهوم النسخ المتماثل متعدد المواقع *(Active-Active)* في *MinIO* لتحقيق التكرارية عبر مراكز بيانات متعددة لضمان توفر البيانات واستمرارية الخدمة [12]

- **النسخ المتماثل أحادي ومتعدد المواقع:** يدعم MinIO النسخ المتماثل على مستويين – داخل الموقع (داخل نفس الكتلة cluster) وعبر مواقع متعددة. على مستوى الكتلة الواحدة، يمكن تشغيل MinIO في وضع **موزع (Distributed Mode)** مع توزيع البيانات عبر عقد متعددة لتحقيق توفر عالٍ؛ يمكن أيضاً تعيين **عامل تكرار Replication Factor** عند إنشاء حاوية (bucket) لتحديد عدد النسخ للبيانات [32] [33] . أما على مستوى المواقع، فميزة **Multi-site Replication** في MinIO تسمح بوجود نشرات متعددة (Active-Active) في مواقع جغرافية مختلفة، حيث يتم نسخ كل كائن مخزن إلى جميع المواقع بشكل شبه متزامن [34] [35] . في حالة تعطل الموقع الأساسي، يمكن تحويل التطبيقات إلى الموقع الثانوي بسرعة (مثلاً عبر موازنة تحميل أو تعديل DNS) ومواصلة الخدمة بنفس البيانات [36] [37] . هذا يضمن **صفر فقد بيانات** عمليًا لأن كل عملية كتابة على الموقع الأساسي تُنسخ فورًا للموقع الآخر [35] . (مع الأخذ بالاعتبار أن النسخ عبر WAN يكون متزامنًا تقريبًا eventual consistency بحيث قد يكون هناك فارق زمني بسيط حسب سرعة الاتصال [35]). الشركات الكبيرة تعتمد هذا النهج عند التعامل مع كميات ضخمة من البيانات لتحقيق RPO وRTO قريبين جدًا من الصفر [38] [39] .

- **النسخ الاحتياطي خارج المنظومة (Offsite Backup):** إلى جانب النسخ الحيّة، يجب الاحتفاظ **بنسخة احتياطية منفصلة** من بيانات MinIO في موقع آخر. إحدى الطرق البسيطة هي استخدام أدوات مثل `mc` [mirror] أو [`mc cp`] لنسخ محتويات حاويات MinIO إلى وجهة خارجية. على سبيل المثال، يمكن إعداد أمر مجدول: `mc mirror local-minio/tenant-bucket remote-minio-backup/tenant-bucket` لعمل تزامن دوري لحاوية مستأجر معين إلى خادم MinIO آخر خارج الموقع. أو استخدام `mc cp` لنسخ المحتويات إلى خدمة تخزين سحابي متوافقة مع S3 [40] [41]. هذه الطريقة تكوّن **نسخة ثابتة دورية** يمكن استعادتها في حال حصول عطب كبير أو حذف غير مقصود في النسخ الحيّة. ينبغي ضبط تكرار النسخ الخارجي (يومي، أسبوعي) بناءً على حجم البيانات وقدرة الشبكة.

- **الإصدارات (Versioning):** كما ذكرنا، تفعيل versioning على مستوى الحاويات يضيف طبقة حماية: إذ يسمح باستعادة إصدارات سابقة من أي ملف تم حذفه أو تعديله عن طريق الخطأ [42] [43]. يُنصح بشدة بتفعيل **Versioning** لحاويات المستخدمين الحرجة. عند التفعيل، كل عملية حذف تؤدي إلى وضع **علامة حذف (delete marker)** بدلاً من إزالة فعلية [44]، ويمكن للمسؤول استعادة الملف بإزالة علامة الحذف. هذا مفيد جدًا لحماية المستخدمين من الأخطاء البشرية (مثل حذف ملف مهم ثم طلب استعادته). ومع ذلك، ستتزايد مساحة التخزين مع الزمن بسبب تراكم الإصدارات، لذا ينبغي استخدام **سياسة دورة حياة (Lifecycle)** لحذف الإصدارات القديمة غير الضرورية بعد فترة معينة [45] [46].

- **الترميز المشتت (Erasure Coding):** يوفّر MinIO خيار **Erasure Coding** كبديل/مكمل للنسخ المتعدد. هذا الأسلوب يقوم بتقسيم كل كائن إلى مقاطع مع حساب بيانات تصحيح الأخطاء (parity) وتوزيعها عبر عدة أقراص أو عقد [47] [48]. الميزة هنا أنه يمكن تحمل فقدان عدة أجزاء (أقراص) مع إمكانية إعادة بناء البيانات، دون الحاجة لتخزين نسخ كاملة مكررة – مما يوفر مساحة مقارنة بالنسخ الكامل. الـErasure Coding مفيد لتحقيق متانة عالية داخل نفس الموقع أو العنقود. لكنه ليس بديلاً عن النسخ الخارجي، إذ أنه لا يحمي من خسارة **جميع العقد** (كالحريق أو حذف منطقي)، لذلك يبقى النسخ إلى موقع آخر ضروريًا.

- **التشفير والحماية:** **يدعم MinIO التشفير للبيانات أثناء التخزين والنقل**. يمكن تفعيل **تشفير البيانات وهي مخزنة (Encryption at rest)** عبر مفاتيح إدارة رئيسية، وكذلك تأمين **الاتصال (Encryption in transit)** عبر SSL [49]. بالنسبة للنسخ الاحتياطية، هذا يعني أن أي بيانات تُنسخ إلى موقع خارجي يمكن أن تُشفر تلقائيًا إذا كانت الوجهة MinIO آخر مفعّل عليه التشفير. أيضًا يمكن استخدام ميزة **التشفير جهة العميل (Client-Side Encryption)** بحيث يقوم عميل النسخ الاحتياطي بتشفير البيانات قبل إرسالها إلى MinIO (MinIO نفسه يدعم ذلك عبر بروتوكول SSE-C حيث يُرسل العميل مفتاح تشفير ولا يحتفظ MinIO بالمفتاح [50]). بغض النظر عن الطريقة، **يجب تخزين النسخ الاحتياطية لملفات العملاء مشفرة** بحيث لو حصل اختراق للمخزن الاحتياطي لا تتعرض خصوصية البيانات للخطر.

- **استعادة granular (لمستأجر محدد):** بفضل تنظيم البيانات في MinIO في حاويات (buckets)، غالبًا ما يُمنح كل مستأجر مساحة تخزين أو حاوية خاصة به. هذا يعني أنه من الممكن استعادة حاوية مستأجر معين من النسخ الاحتياطي دون التأثير على بقية الحاويات. على سبيل المثال، إذا طلب مستأجر استعادة الملفات الخاصة به إلى وضع سابق، يمكننا استرجاع نسخة الحاوية الخاصة به فقط من النسخ الاحتياطي الخارجي إلى بيئة مرحلية، ثم استبدال ملفاته أو إرجاعها إلى MinIO الإنتاجي. وإن كانت ميزة Versioning مفعلة، يمكن ببساطة إعادة تفعيل الإصدارات السابقة للملفات المطلوبة دون الحاجة حتى للذهاب إلى النسخ الخارجي. هذه المرونة في MinIO تجعل **استعادة مستأجر واحد** أكثر سهولة مقارنة باستعادة جزء من قاعدة بيانات مشتركة. ولكن لضمان ذلك، من المهم الحفاظ على **عزل بيانات المستأجرين في حاويات منظمة** وعدم خلط عدة مستأجرين في نفس المسار من دون تمييز.

- **مراقبة MinIO والتنبيه:** يوفر MinIO لوحة تحكم Web UI وأيضًا تكاملاً مع أدوات رصد مثل Prometheus/Grafana لمراقبة حالة الخوادم وعمليات النسخ المتماثل [51] [52]. يجب إعداد تنبيهات لحالة **فشل النسخ المتماثل** أو تأخره. فعلى سبيل المثال، يدعم MinIO إرسال إشعارات لأحداث فشل replication ويمكن إعداد تطبيقات لتلقي هذه الإشعارات وتنبيه فريق العمليات فورًا [53]. كما أن مراقبة سعة التخزين مهمة للتأكد أن النسخ الاحتياطية لا تملأ القرص وتؤدي إلى توقف النظام. أيضًا يفضّل التحقق الدوري من سلامة

البيانات (مثلًا عبر أداة `mc admin integrity` إن وجدت أو من خلال محاولة قراءة ملفات عشوائية من النسخ الاحتياطية).

باختصار، حماية البيانات في MinIO تتطلب **مزيجًا من التكرار الحيّ والنسخ الاحتياطي البارد**. التكرار الحي (Replication) يضمن توفر البيانات حتى عند حصول فشل في خادم أو موقع، أما النسخ البارد (مثل نسخة يومية تُرفع لموقع آخر) فيضمن إمكانية الرجوع لنقطة تاريخية حتى لو حصلت أخطاء من جانب المستخدم أو التطبيق. ويجب ألا ننسى أن استخدام وسائط مختلفة للنسخ (أقراص محلية + سحابة) يحقق قاعدة 3-2-1 ويوفر أمانًا أعلى للملفات.

النسخ الاحتياطي لمحرك البحث المتجه Qdrant

Qdrant هو محرك بحث متجه (Vector Search Engine) يُخزن المتجهات عالية الأبعاد ويُمكّن البحث التقاربي. في سياق تطبيق SaaS، قد يستخدم Qdrant لفهرسة بيانات كل مستأجر (مثلاً كل مستأجر له مجموعة Collection خاصة بالمتجهات التابعة له). على الرغم من أن فقدان بيانات Qdrant قد لا يؤدي إلى فقدان بيانات "معاملات" تقليدية، إلا أنه قد يفقد خدمة البحث أو توصيات الذكاء الاصطناعي الخاصة بالعملاء – وهي جزء مهم من الوظيفة في بعض الأنظمة. لذا ينبغي تضمين Qdrant في خطة النسخ الاحتياطي والتعافي. آليات النسخ الاحتياطي لـQdrant تشمل:

- **ميزة اللقطات (Snapshots) المدمجة:** يوفر Qdrant آلية مضمنة لأخذ **لقطة Snapshot** لمجموعة (Collection) أو حتى لكامل العنقود. هذه اللقطات هي في الأساس **أرشيف tar يحتوي على البيانات وتعريفات التكوين** لتلك المجموعة في لحظة معينة [54]. يتم إنشاء اللقطة عبر واجهة API بسيطة (مثلاً طلب HTTP `POST /collections/{collection_name}/snapshots`) أو باستخدام عملاء Qdrant المتوفرين [55] [56]. اللقطة تشمل كل النقاط (vectors) والحمولات (payloads) في المجموعة، مما يجعلها وحدة نسخ مناسبة لكل مستأجر إذا كان لديه مجموعة منفصلة [57]. في حالة العنقود الموزع، يجب أخذ لقطة على كل عقدة لضمان تغطية جميع أجزاء البيانات [58]. بعد إنشاء اللقطة، يمكن تنزيل ملفها عبر API أيضًا [59] [60]. يُنصح بأخذ **لقطات منتظمة** لمجموعات Qdrant (مثلاً يوميًا أو أسبوعيًا حسب معدل تحديث المتجهات)، وحفظ ملفات اللقطات في موقع تخزين خارجي (يمكن هنا أيضًا استخدام MinIO كمخزن لهذه اللقطات). Qdrant Cloud (الخدمة المُدارة) تتيح جدولة تلقائية للقطات يومية وأسبوعية وشهرية من خلال واجهة الويب [61] [62]، وهذا مؤشر على أهمية إجراء نسخ دوري حتى في خدمات البحث.

- **استرجاع اللقطات (Snapshot Restore):** يدعم Qdrant استعادة البيانات من ملف لقطة بسهولة. يمكن استعادة **مجموعة واحدة** من لقطة عبر طريقتين: إما **أثناء تشغيل الخدمة** باستخدام نهاية API `/collections/{name}/snapshots/recover` حيث يمكن لواجهة API تنزيل ملف اللقطة من عنوان URL أو استخدام ملف محلي [63] [64]، أو **أثناء بدء تشغيل الخادم** بتمرير مسار الملف إلى برنامج Qdrant (خيار `--snapshot` عند تشغيل الخدمة) [65] [66]. الاستعادة تتيح خيار **دمج أو استبدال البيانات الموجودة** عبر ما يسمى **أولوية الاستعادة (snapshot priority)**: يمكن إعطاء أولوية للبيانات الموجودة (replica) أو لبيانات اللقطة (snapshot) أو تعطيل التزامن للسماح باستعادة خاصة [67] [68]. عمليًا، لاستعادة مجموعة لمستأجر محدد دون التأثير على الآخرين، يمكن إنشاء مجموعة جديدة لهذا المستأجر وتحميل اللقطة إليها (أي تغيير الاسم المستهدف عند الاسترجاع حتى لا يكتب فوق المجموعة الحالية) [65] [66]. بعد ذلك قد يقوم فريق الدعم باختبار البيانات المسترجعة ثم تبديل استخدام التطبيق إليها أو دمج التحديثات حسب الحاجة. هذه المرونة في Qdrant تمكن من **استرجاع مستأجر منفرد** أو حتى **استرجاع كامل العنقود** من لقطة، حسب مستوى الحاجة. يجدر ملاحظة أن **التوافقية** مهمة – يجب أن يكون إصدار Qdrant الهدف مطابقًا لإصدار أخذ اللقطة (نفس الإصدار الفرعي) [69] [70] لضمان إمكانية القراءة.

- **التكرار والتوافر العالي:** إذا كان Qdrant يعمل في وضع عنقود مع **عدة نسخ (replicas)** لكل مجموعة، فإن ذلك بحد ذاته يحمي من فقدان البيانات بسبب تعطل عقدة واحدة. فالتكرار يضمن وجود نسخة أخرى من المتجهات على عقدة مختلفة [71] [72]. لكن يجب فهم أنه في حالة خطأ منطقي أو حذف غير مقصود للبيانات عبر واجهة API، فسيتكرر هذا عبر النسخ (لأنها متزامنة)، لذا يبقى **النسخ الاحتياطي التاريخي** ضروريًا للتعافي من الأخطاء البشرية. Qdrant توصي باستخدام النسخ المتماثل للتوافر الفوري (HA) ولكن أيضًا استخدام النسخ الاحتياطية للسيناريوهات الأخرى [73] [74]. أيضًا من الاستراتيجيات المقترحة: **الاحتفاظ**

بمصدر البيانات الأصلي (مثلاً قاعدة بيانات تقليدية أو ملفات) لإعادة الفهرسة في حالات الطوارئ [71]. بمعنى أنه إذا توفرت طريقة لإعادة بناء بيانات Qdrant (كإعادة حساب المتجهات من البيانات الخام)، يمكن اعتبار ذلك كخطة بديلة، وإن كانت بطيئة. لكن عمليًا وجود لقطات دورية يجعل التعافي أسرع بكثير من إعادة الفهرسة الكاملة.

- **النسخ التزايدي في Qdrant:** بشكل افتراضي، كل لقطة Qdrant هي لقطة كاملة لتلك المجموعة. ولكن في بيئات سحابية مُدارة، أشارت وثائق Qdrant Cloud أن النسخ الاحتياطية يمكن أن تكون **تزايدية** للحد من تكلفة التخزين، حيث تحتفظ كل نسخة جديدة فقط بالتغييرات منذ سابقتها (على AWS/GCP) [75] [76]. على VPS عادية، لا يتوفر هذه الميزة بشكل تلقائي، لكن يمكن تحقيق شيء مشابه بواسطة أدوات خارجية (مثلاً حفظ ملفات اللقطة باستخدام أداة deduplication مثل BorgBackup لضمان عدم تكرار المتشابه بينها). على أي حال، نظراً لأن حجم بيانات المتجهات قد يكون كبيرًا لكن معدل التغيير ربما أقل، فإن الاعتماد على **نسخincremental 🚧 فعلي** يمكن أن يوفر مساحة. يجب تقييم ذلك حسب حجم البيانات وعدد المستأجرين – إذا كان لدينا مئات المجموعات لكل مستأجر، ربما من الأفضل أتمتة عملية حذف اللقطات القديمة والاحتفاظ بأحدث N لقطات لكل مجموعة.

- **أمن النسخ الاحتياطي:** ملفات اللقطة تحتوي على بيانات قد تكون حساسة (نظراً لأنها قد تمثل تمثيلات متجهية لبيانات العملاء). لذا مثل بقية المكونات، **ينبغي تشفيرها عند التخزين خارجًا** . إذا تم حفظ اللقطات في MinIO أو تخزين سحابي، يمكن تأمينها عبر التشفير جهة الخادم (SSE) أو تخزينها في حاوية مشفرة. وأيضًا تأكد من ضبط أذونات الوصول بحيث لا يمكن إلا للمسؤولين المعنيين تنزيل تلك اللقطات (مثلاً باستخدام مفاتيح API خاصة أو كلمات مرور).

إجمالاً، التعامل مع Qdrant في خطة النسخ الاحتياطي يتلخص في: **استخدام لقطات دورية، وحفظها خارج خوادم الإنتاج، واستراتيجية واضحة لاستعادة مجموعة واحدة أو عنقود كامل عند الحاجة**. ولأن Qdrant جزء متخصص، يجب تضمين إجراءاته في **كتيب التشغيل عند الكوارث** لفريق العمليات، مع خطوات محددة لكيفية تشغيل الاستعادة وإعادة الخدمة للتطبيق باستخدام البيانات المسترجعة.

النسخ الاحتياطي لوسيط الرسائل RabbitMQ

يعمل **RabbitMQ** كوسيط رسائل يربط بين مكونات النظام، وضمان وصول الرسائل (مثلاً أوامر أو أحداث بين الخدمات) جزء أساسي من سلامة المنصة. ومع ذلك، تختلف استراتيجية حماية البيانات في RabbitMQ عن قواعد البيانات أو التخزين، لأن **الرسائل بطبيعتها عابرة** ؛ حيث يُفترض استهلاكها ومعالجتها فورًا أو خلال وقت قصير. لذلك يركز التعافي هنا على أمرين: **حماية بنية النظام (التعريفات)** و **ضمان عدم فقدان الرسائل المهمة في حال فشل العقد**. النقاط المهمة تشمل:

- **نسخ تعريفات RabbitMQ (Definitions):** التعريفات تشمل كل بيانات إعداد RabbitMQ مثل **المستخدمين، الصلاحيات، المضيفات الافتراضية (vhosts)، الطوابير، التبادلات (exchanges)، الربط (bindings)، السياسات...إلخ** [77] [78]. هذه البيانات ثابتة نسبيًا ويمكن **تصديرها إلى ملف JSON** باستخدام أدوات RabbitMQ المدمجة. مثلًا:

```
rabbitmqctl export_definitions /path/to/defs.json
```

سينتج ملفًا يحتوي كل التعريفات. هذا الملف يجب الاحتفاظ به كنسخة احتياطية (ربما مع كل تغيير كبير في البنية أو بشكل دوري كل 8 ساعات كما تفعل بعض الخدمات المدارة [79] [80]). في حال حدوث كارثة تستدعي إنشاء خادم RabbitMQ جديد، يمكن ببساطة **استيراد التعريفات**:

```
rabbitmqctl import_definitions /path/to/defs.json
```

وبذلك تُعاد تهيئة جميع التبادلات والطوابير والمستخدمين كما كانت [81] [82] . هذا يغطي استعادة الهيكلية بحيث تتطابق البيئة الجديدة مع القديمة. يُذكر أن **CloudAMQP** (خدمة RabbitMQ السحابية) تحتفظ آليًا بملف التعريفات كل 8 ساعات مما يبين أهمية ذلك في خططهم [79] . أيضًا التناغم بين نسخة التعريفات وبيانات التطبيقات مهم للحفاظ على استمرارية العمل بسرعة عند Disaster Recovery. تجدر الإشارة إلى أن تعريفات RabbitMQ في حال عمله كعنقود تكون مكررة على كل عقدة، وبالتالي يمكن تصديرها من أي عقدة [83] .

- **التوافر العالي للرسائل (HA) بدلًا من النسخ الاحتياطي التقليدي:** بالنسبة **للرسائل المخزنة** في RabbitMQ، عادةً لا يتم أخذ نسخ احتياطية دورية منها بالمعنى التقليدي (حيث أن الرسائل ديناميكية وقديمة بعد استهلاكها). عوضًا عن ذلك، يُحقق الأمان عبر **بنية عالية التوافر**:
- تشغيل RabbitMQ كعنقود من عدة عقد (nodes cluster). هذا يضمن أنه إذا تعطل خادم واحد، يستمر الآخرون في الخدمة دون فقدان عام [84] [85] .

- استخدام **الطوابير المتكررة (Mirrored Queues)** أو **طوابير النصاب (Quorum Queues)**: في حالة الطوابير التقليدية يمكن تطبيق سياسة **mirroring** بحيث يتم نسخ كل رسالة إلى عُقد أخرى؛ أما طوابير Quorum (المضافة في الإصدارات الحديثة) فتحتفظ بسجل WAL موزع وتقدم موثوقية أعلى وافضل لإستبدال الـ mirrored القديمة [86] [87] . باستخدام هذه الآليات، أي رسالة تصل إلى طابور حرج ستكون متوفرة على أكثر من عُقدة، وبالتالي تعطل عقدة منفردة لن يؤدي لضياع الرسالة. هذا الأسلوب أشبه بالنسخ **الآني المستمر** بدل النسخ الاحتياطي الدوري التقليدي. يجدر التخطيط لأنواع الطوابير: فالطوابير المتكررة تزيد الحمل على النظام (لكونها تكتب على عدة عُقد) ولكنها ضرورية للرسائل ذات الأهمية العالية.

- **الكتابة المتزامنة إلى القرص (Sync Disk Writes):** بشكل افتراضي، RabbitMQ يستخدم كتابة غير متزامنة للرسائل المتينة على القرص (يعتمد على نظام التشغيل لكتابتها في الخلفية). في سيناريوهات حرجة، يمكن تفعيل الكتابة المتزامنة (بالسياسات) لبعض الطوابير بحيث لا تعتبر الرسالة مستلمة إلا بعد تأكيد كتابتها فعليًا إلى القرص [88] [89] . هذا يقلل احتمال فقد رسائل عند انقطاع التيار مثلًا، لكنه يؤثر على الأداء لذا يُفعّل انتقائيًا (مثال: وضع سياسة SyncWrite للطوابير الحرجة).

- **النسخ الاحتياطي للرسائل – استثناءات وحالات خاصة:** على الرغم من أن القاعدة هي ضمان التوافر عبر HA بدلًا من نسخ الرسائل، قد توجد حالات تحتاج فيها حفظ الرسائل غير المستهلكة خارج RabbitMQ (مثلاً عند ترقية كبيرة أو إيقاف مخطط، أو لأغراض التدقيق). يوجد أدوات مثل **RabbitMQ Streams** (لتسجيل متواصل للرسائل) أو حلول خارجية مثل أداة **RabbitIO** التي طورتها شركة Meltwater والتي **تسحب الرسائل إلى ملف** ومن ثم تعيد دفعها عند الحاجة [90] . هذه الأدوات يمكن استخدامها لإفراغ طوابير إلى تخزين وسيط (مثل JSONLines أو CSV) كنسخة احتياطية يدوية ثم إعادة إدخالها. لكنها ليست عملية للاستخدام اليومي، بل ربما لإجراءات انتقالية أو تحقيق في مشكلة. توصي وثائق RabbitMQ الرسمية بأنه إن أردت نسخ الرسائل المخزنة، عليك **إيقاف الخدمة وأخذ نسخة من مجلد البيانات** (الذي يحتوي رسائل الطوابير) [91] [92] . هذا ممكن تقنيًا – فمجلد `/var/lib/rabbitmq/mnesia/` يحوي ملفات تخزين الرسائل للطوابير الكلاسيكية (`msg_stores`) وطوابير النصاب (`quorum`) [93] [94] . ولكن القيام بذلك على عقدة شغّالة غير محبذ إطلاقًا لأن الرسائل في تدفق مستمر وقد تنتج نسخة غير متسقة [91] . إذا اضطررت لذلك، **يجب إيقاف جميع عُقد العنقود** لتجنب تغييرات أثناء النسخ [95] . لهذا السبب، يفضل اعتماد HA كما ذكرنا لتجنب الحاجة لهذا السيناريو المعقد.

- **التكرار الجغرافي (DR Sites):** إذا كانت متطلبات الكوارث تشمل تعطل مركز بيانات كامل، فيجب التفكير في وجود **وسيطة رسائل احتياطية في موقع آخر** . RabbitMQ لا يدعم نسخ العنقود عبر WAN بشكل مباشر (أي لا يوجد خاصية cluster stretching officially)، ولكن يمكن استخدام **الفيدرation أو الـShovel**. على سبيل المثال، يمكن إعداد **RabbitMQ Federation** بين عنقود الإنتاج وعنقود آخر في مركز بيانات مختلف بحيث يتم **نسخ الرسائل المختارة** (عادة من exchanges محددة) إلى العنقود الآخر [96] . هذا يشكل نوعًا من **Hot Standby** : العنقود الثاني يستقبل نفس الرسائل في الوقت الحقيقي تقريبًا [97] [98] . في حال كارثة للمركز الأول، يمكن تحويل المستهلكين إلى العنقود الثاني. خدمات مثل CloudAMQP توفر إرشادات لمثل هذا الإعداد (استخدام federation على مستوى exchange مهم للأولويات القصوى) [98] [99] . بالمقابل، هناك خيار **Cold Standby** أبسط: الاحتفاظ فقط بنسخة التعريفات وإنشاء عنقود جديد عند اللزوم

واستقبال الرسائل الجديدة عليه [100] [101] ؛ سيكون هناك فقد رسائل قيد النقل حين وقوع الكارثة لأن CloudAMQP مثلاً لا تحتفظ بنسخ للرسائل نفسها [102] . القرار بين hot/cold standby يعود لمتطلبات العمل (مدى تحمل فقدان بعض الرسائل مقابل التعقيد والكلفة). للأهمية القصوى، hot standby مع federation يكون الحل لكن يجب تحمل مضاعفة حركة الرسائل والتعقيد التشغيلي في إبقاء كلا العنقودين متزامنين (بما في ذلك **مزامنة التعريفات دوريًا** بينهما) [103] [104] .

- **مراقبة RabbitMQ والتنبيهات:** من الضروري مراقبة **صحة العنقود** (عدد العُقد العاملة) و**طول الطوابير** ومستوى **استهلاك القرص** في RabbitMQ. استخدام أدوات مثل **Prometheus مع Exporter لـRabbitMQ** وجرافانا يساعد في إنشاء لوحات قياس. قم بضبط تنبيهات في حال سقوط أي عُقدة أو ارتفاع غير معتاد في طول طابور (ما قد يشير إلى تعطل مستهلك أو تراكم رسائل قد يؤدي لاستنفاد الموارد). أيضًا مراقبة السياسات (مثل هل ما زالت الطوابير الحرجة تحت mirroring أم لا) مفيدة. بالنسبة للفشل، RabbitMQ نفسه لا يرسل تنبيهات خارجية بشكل افتراضي، لكن يمكن التكامل مع **Shovel Plugin** أو غيره لتمرير تنبيهات إلى نظام خارجي. على الأقل، تأكد أن **فشل عملية تصدير النسخ الاحتياطي للتعريفات** سيبلغ الفريق (مثلاً إن كان هناك Cron job يصدر التعريفات، اجعل إخفاقه يرسل بريدًا أو إشعارًا).

- **اختبارات الاستعادة والفشل:** كما في بقية المكونات، يجب **اختبار سيناريوهات الفشل**. مثلاً، جرّب إغلاق عقدة RabbitMQ بشكل مفاجئ وتحقق من أن العنقود يستمر بالعمل وأن أي طوابير متكررة لم تفقد رسائل (يمكن إجراء اختبار ببث عدد من الرسائل على طابور متكرر ثم قتل عقدة ورؤية إن كانت الرسائل متاحة على العقدة الأخرى). أيضًا جرّب **استيراد ملف تعريفات** على عنقود جديد في بيئة تجريبية لترى إن كان يعيد إنشاء كل المكونات بشكل صحيح [105] . هذه الاختبارات تكشف أي اختلافات في الإصدارات أو المشاكل في ملفات النسخ الاحتياطي قبل وقوع حدث حقيقي.

خلاصة القول، في RabbitMQ نهتم أكثر **بالتوافر الآني** من النسخ التاريخي. النسخ الاحتياطي هنا يتعلق بالتكوين، أما الرسائل فنضمنها عبر التكرار الفوري. ومع ذلك يجب عدم إغفال أن **فقدان الرسائل يمكن التعامل معه أيضًا على مستوى التطبيق** : فالتطبيقات الحساسة تصمم أحيانًا لضمان **عدم ضياع العمل عند ضياع رسالة** – كأن يقوم المرسل بإعادة المحاولة إذا لم يتلق تأكيدًا، أو أن يكون المستهلك idempotent بحيث لو استقبل رسالة مكررة لا مشكلة. هذه المستويات الإضافية من التحمل تجنب الاعتماد الكلي على أن الرسائل محفوظة 100% طوال الوقت (فقد تفقد في أسوأ الأحوال). لكن مع تطبيق ما سبق، يكون نظام الرسائل قويًا وقادرًا على تجاوز معظم مشاكل الفقدان.

الأدوات المفتوحة المصدر والخدمات الداعمة للنسخ الاحتياطي (في بيئة VPS)

في بيئة خادم خاص (VPS) خارج إطار المنصات السحابية الكبيرة، يكون العبء أكبر على المطور في بناء حلول النسخ الاحتياطي. لحسن الحظ هناك العديد من **الأدوات المفتوحة المصدر** التي يمكن أن تساعد في أتمتة وإدارة النسخ الاحتياطي بمختلف أنواعه، وتتكامل مع التقنيات التي لدينا:

- **أدوات نسخ احتياطي عامة للملفات والأنظمة:** من أبرزها:
- **Restic:** أداة حديثة وسريعة للنسخ الاحتياطي الآمن. تمتاز بأنها تدعم التشفير بشكل افتراضي، والرفع إلى عدة وجهات (محلية أو عبر SSH أو S3/MinIO). تقوم Restic بعمل deduplication (عدم تكرار البيانات) مما يجعلها فعّالة للمساحات الكبيرة. يمكن استخدامها لنسخ مجلدات النسخ الاحتياطية (مثل مجلدات MongoDump أو لقطات Qdrant) إلى مستودع خارجي مشفر.
- **BorgBackup:** أداة قوية للدعم الاحتياطي مع ضغط وتشفير وتحرّي التكرار أيضًا. Borg يمكن أن تعمل عبر SSH بسهولة لذا قد تُستخدم لنفس سيناريوهات Restic. كلاهما يعتمد على سطر الأوامر مما يسمح بجدولتها عبر cron.

- **Duplicity:** أداة قديمة نسبياً لكنها موثوقة، تدعم النسخ **التزايدي** مشفّر (تعتمد على GnuPG). تقوم بإنشاء أرشيفات tar مشفرة ويمكنها التخزين على خدمات مثل S3 أو حتى Google Drive [106] [107] . قد تكون ملائمة لرفع النسخ إلى التخزين السحابي مباشرة.
- **Rclone:** ليس أداة نسخ احتياطي بقدر ما هو أداة **مزامنة ملفات** تدعم بروتوكولات عديدة منها S3 وWebDAV وغيرها [108] [109] . يمكن استخدامها لجدولة رفع ملفات النسخ الاحتياطي (مثل ملفات JSON لتعريفات RabbitMQ، أو أرشيفات mongodump) إلى مواقع متعددة بشكل سهل.
- **Bacula/Amanda:** أنظمة نسخ احتياطي شبكية متكاملة. تتميز هذه بأن لديها خادم عميل وهيكلية شاملة للنسخ من عدّة عملاء إلى مخزن مركزي مع جداول وجدولة معقدة [110] [111] . قد تكون ثقيلة للإعداد لكنها مفيدة إن كان لدينا بنية تحتية كبيرة ونريد إدارة مركزية للنسخ. في سياقنا (عدة تطبيقات على VPS)، قد يكون استخدام أدوات أخف وزن مثل التي سبق ذكرها أكثر بساطة.

من الجدير بالذكر أن بعض هذه الأدوات **توفر مزايا مفيدة** مثل: الضغط والتشفير المدمج (Restic, Borg, Duplicity كلها تفعل ذلك)، دعم النسخ التزايدي على مستوى الملفات (Duplicity, Borg)، التكامل مع جداول Cron بسهولة. **عدة فرق في الصناعة تستخدم مزيجًا من هذه الأدوات** . على سبيل المثال، يقومون باستخدام BorgBackup لأخذ نسخ احتياطية مشفرة من ملفات قاعدة البيانات بشكل يومي وحفظها على خادم آخر. أو استخدام Restic للنسخ إلى خدمة Backblaze B2 (تخزين سحابي منخفض التكلفة). هذه الأدوات موثوقة ومجربة على نطاق واسع [112] (تذكر المصادر أن Restic وBaculag Velerog Duplicityg Amandag من أكثر الأدوات شيوعًا [112]).

- **أدوات متخصصة بحسب الخدمة:**
- بالنسبة لـ**MongoDB**: بالإضافة إلى Percona Backup المذكور، هناك **Mongodb Atlas Backup** (في حالة عدم استخدام Atlas لدينا، لكن ممكن الاستلهام منه)، وأيضًا **Wal-G** التي تدعم MongoDB لإجراء نسخ احتياطية تعتمد على سجل الكتابة (Write Ahead Log) تشبه pg_wal في PostgreSQL. Wal-G مفتوحة المصدر وتُستخدم عادة مع PostgreSQL، لكنها أعلنت دعم MongoDB وتتيح نسخًا مستمرة على مستوى الحزم الثنائية للبيانات (سريعة جدًا) مع الاحتفاظ بلقطات [113] . الاستفادة من Wal-G تتطلب فهم عميق وإعداد جيد لكنها تستحق النظر للمستقبل إن كان حجم البيانات يزداد كثيرًا.
- بالنسبة لـ**RabbitMQ**: توجد أدوات مثل **Lapin** أو **RabbitDump** غير الرسمية، ولكن كما بينا النسخ عبر التصدير المدمج كافٍ للتعريفات.
- بالنسبة لـ**MinIO**: الأداة الأساسية هي `mc` (MinIO Client) فهي بحد ذاتها مفتوحة المصدر وتدعم كل العمليات (cp, mirror) اللازمة. أيضًا **MinIO Operator** للكوبيرنيتس يوفر جدولة نسخ، لكن على VPS قد لا نستخدم Kubernetes. إن تم استخدام Docker مثلاً، يمكن التفكير في استخدام **Velero** مع plugin لمجلدات الـVolumes. ولكن Velero أساسًا مخصص لكوبيرنيتس [114] .

- بالنسبة لـ**Qdrant** : حاليًا الطريقة المباشرة هي عبر API snapshots كما ذكرنا. يمكن كتابة **سكربت Python** باستخدام `qdrant-client` يقوم بأخذ لقطات لجميع المجموعات بشكل مجدول (باستخدام مكتبة جدولة مثل APScheduler)، ثم يحفظ الملفات إلى MinIO. هذا نوع من "الأداة" المخصصة لكن باستخدام مكونات Qdrant الرسمية.

- **الخدمات الخارجية:** إن كنا نرغب بحلول جاهزة دون إدارة ذاتية بالكامل، هناك مزودون متخصصون في النسخ الاحتياطي للبيئات ذاتية الاستضافة. مثلاً:

- خدمات مثل **Acronis Backup** أو **Veeam** (مع أنهما حلول تجارية غير مفتوحة المصدر) تدعم أخذ نسخ احتياطية من خوادم VPS كاملة (صورة آلة) أو ملفات محددة. Veeam مثلاً يدعم النسخ إلى MinIO (Object Lock) لضمان نسخ ضد الفدية [115] [20] . لكنها قد تكون باهظة الثمن.
- خدمة **CloudAMQP** تم ذكرها لحالة RabbitMQ – يمكن استخدامها كحل DR، حيث تبقي عنقود RabbitMQ ثانوي جاهز. لكن ذلك يعتمد على سحابة خارجية.
- بعض شركات الاستضافة تقدم خدمة **Snapshots دورية للـVPS** نفسه. هذه أيضًا شكل من أشكال النسخ الاحتياطي (صورة شاملة للنظام). يمكن الاستفادة منها كطبقة أخيرة في حالة فشل كل شيء، بحيث نعيد تشغيل VPS من لقطة حديثة. لكن يجب عدم الاقتصار عليها لأنها لا تفرق بين المستأجرين ولا تسمح باستعادة جزء محدد بسهولة.

باختصار، الأدوات المفتوحة المصدر وفيرة، ويمكن المزج بينها: مثلًا **استخدام** `mongodump` **مع ضغط Snappy ورفعه بواسطة Restic إلى مستودع مشفر**، أو **أخذ لقطات Qdrant وحفظها بواسطة Rclone إلى مخزن بعيد**. المفتاح هو أتمتة هذه العمليات (باستخدام Cron أو أنظمة CI/CD) ومراقبتها كما سنوضح تالياً.

جدولة النسخ الاحتياطي والتكرار وتخزين النسخ في مواقع متعددة

وضع **جدول زمني واضح** للنسخ الاحتياطي ضروري لضمان الالتزام وعدم النسيان. يجب التخطيط لتواتر النسخ بناءً على **هدف نقطة الاستعادة (RPO)** المقبول. إن كان RPO لخدمتنا هو 24 ساعة (أي يمكننا تحمل فقدان بيانات آخر يوم كحد أقصى في أسوأ الأحوال)، فيجب أن تكون النسخة الاحتياطية **يومية على الأقل**. إن كان بعض العملاء يطلبون RPO أقصر (ساعات مثلاً)، قد نحتاج لنسخ مستمر (Continuous Backup) لبعض البيانات. عناصر الجدولة تشمل:

- **نسخ يومية (أو أكثر تكرارًا) للبيانات الأساسية:** عادة تُختار ساعة خفيفة الحمل (مثلاً بعد منتصف الليل) لإجراء النسخ اليومية. يمكن إعداد **مهام Cron** على الخوادم تقوم بتنفيذ سلسلة النسخ: مثلاً تشغيل `mongodump` كل يوم 02:00، وتشغيل سكربت لأخذ لقطات Qdrant 02:30، وهكذا مع فارق زمني بسيط لتجنب تزامن قد يرهق I/O. يجب كتابة سجلات (Logs) لكل عملية لمراجعة نجاحها أو فشلها.

- **نسخ أسبوعية/شهرية كاملة:** بالإضافة إلى النسخ التزايدية اليومية، ينصح عادةً بأخذ **نسخة كاملة أسبوعية** على الأقل [21]. مثلاً مساء كل جمعة نسخة MongoDB كاملة، وأيضًا تفريغ كل ملفات MinIO إلى أرشيف. هذه النسخ الكاملة الأسبوعية يمكن الاحتفاظ بها لمدة أطول (عدة أشهر). أيضًا ربما نسخة شهرية كاملة تحفظ لفترة سنوية للاحتياط (بعض المؤسسات تحتفظ بنسخة نهاية كل سنة على مدى 5 سنوات مثلاً لأغراض أرشيفية).

- **التكرار (Retention):** حدد بدقة كم من الوقت ستحتفظ بكل فئة من النسخ. الأمثلة النموذجية: احتفظ بالنسخ اليومية لآخر 7 أيام، والنسخ الأسبوعية لآخر 4 أسابيع، والنسخ الشهرية لـ12 شهر. طبق ذلك على كل نوع من البيانات. التطبيق العملي: يمكن لسكربت النسخ بعد انتهائه حذف الملفات الأقدم من مدة معينة أو نقلها إلى تخزين أرشيفي أرخص. الهدف هو تحقيق توازن بين **توفر نقاط استعادة متعددة** وعدم تكديس بيانات قديمة جدًا تستهلك التخزين بدون داع. في سياق متعدد المستأجرين، قد يُطلب الاحتفاظ ببعض النسخ المتعلقة بعميل مهم لفترة أطول بناءً على اتفاقيات خدمة. يجب مراعاة ذلك إن وجد.

- **تعدد مواقع التخزين:** لا يكفي خزن جميع النسخ في نفس الـVPS أو حتى نفس المركز. نفّذ **نسخة موازية إلى موقع آخر**. أمثلة: بعد أخذ نسخة محليًا، يستخدم سكربت `rclone` لرفعها إلى **خادم آخر** (VPS ثانوي أو NAS في مكتب آخر) وأيضًا إلى **تخزين سحابي** (مثلاً خدمة مثل Backblaze B2 أو AWS S3 أو Azure Blob حسب المتاح). وجود نسختين خارجيين في موقعين مختلفين يحقق مستوى عالي من الأمان (هذا الـ3-2-1: نسخة إنتاج + محلي + خارجي [3] [4]). أيضًا إن كان لديك أكثر من مركز بيانات خاص، يمكن تبادل النسخ بينهم (Site A يخزن نسخة من Site B والعكس). بالنسبة لنسخ MinIO، فإن **التكرار المتعدد المواقع** الذي ذكرناه يمكن اعتباره طريقة النسخ المستمر إلى موقع آخر في الوقت الحقيقي [12] [13]، ولكنه لا يغني عن أخذ نسخة ثابتة قديمة (لأن التكرار آنيا لن يحمي من أخطاء منسوخة كما هي). لذا ربما الأفضل هو الجمع: موقعان نشطان + نسخة أرشيفية خارجية.

- **التنسيق بين المكونات:** يجب تنسيق الجدول بحيث **تُربط النسخ بين المكونات**. مثلاً لو أخذنا نسخة MongoDB الساعة 2:00 ونسخة MinIO الساعة 3:00، هناك فجوة ساعة بينهما قد تتغير فيها البيانات. في حالة الحاجة لاستعادة **نقطة زمنية موحدة**، هذا قد يسبب عدم تطابق. للتغلب على ذلك، يمكن:

- إما تقريب أوقات النسخ جدًا لجعل الفارق ضئيل (خلال دقائق).
- أو إيقاف التطبيق أثناء نافذة النسخ (Downtime قصير مخطط له) لضمان عدم حدوث تغييرات لحظية – وهذا غير محبذ في SaaS عالي التوفر إلا إذا كان الفارق دقائق في ساعات الفجر.

- أو الاعتماد على **علامات زمنية** : مثلاً تسجيل timestamp قبل بدء النسخ، ثم استخدامه كمرجع. MongoDB بإمكانه بدعم oplog استعادة حتى علامة زمنية معينة. وMinIO عبر versioning يمكننا استخراج إصدارات الملفات عند تلك اللحظة. هذه الطرق معقدة لكنها تحقق نقطة اتساق. معظم الأنظمة تقبل مقدار ضئيل من عدم التطابق (مثل ربما تُفقد رسالة غير مهمة أو ملف تم رفعه خلال الدقيقة تلك)، لكن الأفضل تقليله قدر الإمكان.

- **التوثيق والآتمتة:** اجعل الجدول ورسائل نجاح/فشل كل مهمة واضحًا ومتاحًا للفريق. يمكن استخدام أدوات الجدولة المتقدمة مثل **Job schedulers** (مثلاً Jenkins Jobs أو Kubernetes CronJobs إذا استخدمت K8s، أو حتى أدوات مثل Ansible AWX) لإدارة مهام النسخ بمرونة أكثر من Cron التقليدي. هذه الأدوات قد تسمح بترتيب المهام مع شرط إتمام مهمة قبل بدء الأخرى، مما يساعد في تنسيق نسخ متعددة المكونات. المهم أن تكون عملية النسخ "بلا لمسة بشرية" after setup، لضمان انتظامها.

تشفير وحماية النسخ الاحتياطية

أمان النسخ الاحتياطية لا يقل أهمية عن أمان البيانات الأصلية. في الواقع، كثيرًا ما تُستهدف النسخ الاحتياطية من قبل المخترقين أو برمجيات الفدية، لأنها قد تبدو أقل حماية أو قد تكون مخزنة على خوادم منفصلة. لضمان حماية النسخ الاحتياطية:

- **التشفير الشامل (End-to-End Encryption):** يجب أن تكون أي بيانات يتم نسخها **مشفرة** سواء أثناء انتقالها عبر الشبكة أو أثناء تخزينها على الوسيط الاحتياطي. أثناء النقل، استخدام بروتوكولات آمنة (HTTPS, TLS for S3, SSH لنقل الملفات) أمر أساسي [49] . أثناء التخزين، هناك خيارات:
- **تشفير جهة العميل:** حيث يتم تشفير البيانات قبل إرسالها إلى التخزين. مثال: Duplicityg Restic يقومان بهذا؛ إذ يُخزَّن فقط البيانات المشفرة والمضغوطة على الوجهة، ولا يمكن قراءتها إلا بمفتاح/كلمة مرور يعرفها المسؤول. هذا مثالي لأن حتى لو تم اختراق مخزن النسخ، تبقى البيانات غير قابلة للفهم بدون المفتاح.
- **تشفير جهة الخادم:** إذا كان التخزين يدعم التشفير الداخلي (مثل MinIO يدعم SSE-S3/Master key، أو إذا خزّنا في AWS S3 مع تمكين SSE-KMS)، فهذا يضيف طبقة حيث البيانات على القرص في المخزن مشفرة. لكن يجب ملاحظة أن المسؤول عن المخزن قد يستطيع الوصول للبيانات إذا تحكم بالمفاتيح، لذا التشفير من جهة العميل أكثر أمانًا.

- في حال النسخ على أجهزة ملموسة (قرص خارجي أو شريط tape) يجب أيضًا تشفيرها بواسطة برامج قبل تخزينها. استخدام أدوات مثل gpg لتشفير الملفات الكبيرة ممكن ولكنه أقل ملاءمة من حلول النسخ الاحتياطي المضمنة التشفير.

- **إدارة المفاتيح:** جانب مهم من التشفير هو الحفاظ على أمان مفاتيح التشفير/كلمات المرور ذاتها. يجب تخزين مفاتيح التشفير في مكان آمن ومع وجود نسخ احتياطية منها أيضًا (يفضّل في مدير كلمات سر أو خزانة مفاتيح داخلية). كثير من قصص الفشل تشمل وجود نسخة احتياطية مشفرة ولكن **فقدان كلمة المرور** مما جعلها عديمة الجدوى. لذا تأكد من توثيق معلومات فك التشفير بشكل آمن يمكن الوصول إليه من قبل الأشخاص المصرح لهم في حالات الطوارئ.

- **التحصين ضد برمجيات الفدية (Ransomware):** لحماية النسخ من الحذف أو التشفير الخبيث، ينصح بتطبيق مبدأ **immutability** لبعض النسخ. مثلًا، إذا تم خزن النسخ في MinIO أو نظام يدعم **قفل الكائن Object Lock** ، يمكن ضبط سياسة جعل النسخ غير قابلة للحذف/التعديل لفترة محددة. العديد من حلول التخزين توفر ما يسمى "حاوية مقفلة" حيث حتى من يمتلك صلاحيات إدارية لا يستطيع حذف البيانات خلال مدة القفل. مثلاً، عند استخدام Backblaze B2 أو AWS S3 يمكن تفعيل خاصية **WORM (Write Once Read Many)** للنسخ الاحتياطية. هذا يمنع المهاجم الذي يحصل على الوصول من ببساطة حذف كل النسخ. بالطبع هذا يحتاج موازنة مع القدرة على إدارة التخزين (لا تريد قفل كل شيء للأبد). حل آخر هو الاحتفاظ بنسخة "عزل" offline – مثل نسخها إلى قرص صلب يفصل عن الشبكة (air-gapped).

- **التحكم بالوصول (Access Control):** تأكد أن مخازن النسخ الاحتياطي مستقلة عن حسابات المستخدمين اليومية. مثلاً، لا تترك خادم النسخ الاحتياطي موصولاً بنفس الشبكة المفتوحة للتطبيق بدون قيود. اجعل الوصول إليه عبر شبكة إدارة منفصلة أو VPN. امنح أذونات أقل ما يمكن: الحساب الذي يكتب النسخ إلى التخزين السحابي يجب ألا يستطيع حذفها إن أمكن (بعض الخدمات تسمح بفصل صلاحيات put عن delete). وفي MinIO يمكنك إنشاء **سياسة IAM** للحاوية الاحتياطية بحيث فقط يسمح بالكتابة وقراءة النسخ وليس حذفها إلا بتدخل خاص.

- **سلامة البيانات (Integrity):** بالإضافة للتشفير، ينبغي التأكد من سلامة النسخ وعدم تعرضها لتحريف متعمد أو غير متعمد. كثير من أدوات النسخ ينشئ **مجموعات تحقق (Checksums)** لكل ملف. يجب الاستفادة منها عبر **التحقق الدوري** (مثل أمر `restic check` الذي يفحص سلامة المستودع ويكشف أي أجزاء تالفة). إذا وجدت مشكلة سلامة، يجب التحقيق فورًا: هل هو قرص تالف؟ هل حصل اختراق وحُرّفت البيانات؟ ومعالجة الأمر (ربما بإعادة إنشاء نسخة سليمة من المصدر). هذا جزء من الحماية لأنه يضمن أنه عندما نحتاج النسخة سنجدها صالحة [116] [31] .

- **السجلات والتدقيق:** احتفظ بسجلات عمليات النسخ الاحتياطي وعمليات الاستعادة أيضًا. راقب هذه السجلات للتعرف على أي نشاط غير معتاد (مثل محاولة استعادة غير مصرح بها أو تعديل في جدول النسخ). وجود **تنبيه عند حدوث نسخ احتياطي خارج الجدول** قد يدل على محاولة مهاجم أخذ بيانات. أيضًا بعض الأنظمة قد تخزن بيانات حساسة قانونيًا (PII)، فالتدقيق مهم لمعرفة من ومتى تم الوصول لنسخ البيانات.

إجمالًا، يجب النظر للنسخ الاحتياطية على أنها **كنز ثمين يجب حمايته** بقدر حماية النظام الإنتاجي نفسه. فكّر ماذا سيحدث إن تمكن شخص من الحصول على ملفات نسخ MongoDB مثلًا – سيملك كل بيانات عملائك! لذلك فالتشفير والحماية ليست أمورًا تجميلية بل ضرورية للغاية في سياق SaaS حيث البيانات متعددة العملاء وحساسة.

آليات استرجاع بيانات مستأجر محدد عند الطلب

من السيناريوهات العملية في SaaS متعددة المستأجرين هو أن **يتعرض أحد المستأجرين لمشكلة تتطلب استعادة جزئية** دون التأثير على باقي النظام. ربما قام عميل ما بحذف كمية من بياناته عن طريق الخطأ ويريد استعادتها، أو حصل فساد في بيانات ذلك العميل تحديدا (دون المساس بالآخرين). تحقيق ذلك يتطلب تخطيطًا منذ تصميم النسخ الاحتياطي لكيفية الفصل بين بيانات المستأجرين. فيما يلي الآليات لكل طبقة:

- **استرجاع بيانات مستأجر من MongoDB:**
إذا كان كل مستأجر في **قاعدة مستقلة** أو مجموعات (collections) مستقلة، يكون الأمر سهلاً نسبيًا. يمكن إجراء **استعادة محددة لقاعدة أو مجموعة واحدة** باستخدام أدوات MongoDB. على سبيل المثال:
- بواسطة `mongorestore` يمكنك تحديد اسم قاعدة محددة أو حتى مجموعة محددة من ملف النسخ الاحتياطي ليتم استعادتها [117] . لو كانت نسختك الاحتياطية تشمل جميع القواعد، فيمكن عند الاسترجاع فلترة قاعدة المستأجر المستهدف فقط. هذا سيعيد بيانات ذلك المستأجر لوحدها. طبعًا يُنصح بأن تتم الاستعادة إلى **خادم MongoDB مؤقت** (isolated) أولاً للتأكد من صحة البيانات، ثم نسخها إلى الإنتاج (إما عبر export/import أو من خلال أدوات sync).
- إن كان المستأجرون مشتركين في نفس المجموعات (تمييز بحقول معرف tenant)، تصبح العملية أدق. إحدى الطرق: **استخدام استعلام** لاسترجاع وثائق ذلك المستأجر. مثلاً: استعادة نسخة كاملة للبيانات إلى قاعدة مؤقتة، ثم استخراج جميع الوثائق حيث tenant_id = X، ثم إدخالها في قاعدة الإنتاج (مع التعامل بحذر مع التعارضات أو المفاتيح). يمكن هنا الاستفادة من أدوات مثل `mongoexport` مع query أو سكربت برمجي. هذه العملية معقدة وتستلزم مراجعة. لذلك يفضّل منذ التصميم فصل البيانات قدر الإمكان لتسهيل مثل هذه السيناريوهات.

- MongoDB Ops Manager (إن استُخدم) لديه ميزة **Queryable Backup** تسمح باستخراج مجموعة فرعية من البيانات من النسخة الاحتياطية بدون استعادة كاملة، وهذا مثالي لمثل هذا السيناريو، لكنه تقنية متقدمة غالبًا لا تتوفر في البيئة المفتوحة المصدر بدون تكلفة.

- **استرجاع ملفات مستأجر من MinIO:**
كما أسلفنا، لو كان لكل مستأجر حاوية مستقلة أو على الأقل مسار مستقل (مثل `tenant-id/`)، فيمكن استعادة تلك الأجزاء تحديدًا. السيناريوهات:

- إذا كان **الإصدارات Versioning** مفعّلًا، غالبًا حل المشكلة يكون مباشرًا عبر إظهار الإصدارات القديمة وإزالة علامة الحذف للملفات المطلوبة أو استرجاع النسخ السابقة منها. يمكن للمسؤول القيام بذلك عبر لوحة MinIO أو `mc`. مثلًا: `mc cp myminio/tenant-bucket/file.jpg\?versionId=XYZ local/tmp/file.jpg` لجلب نسخة قديمة ثم إعادة رفعها كملف جديد.
- إذا كانت هناك نسخة احتياطية منفصلة (كاملة) للحاوية، فيمكن إنشاء **مثيل مؤقت** من MinIO ونسخ بيانات المستأجر إليه، ثم نسخ الملفات المطلوبة إلى الإنتاج. أو ببساطة، استخدام `mc` لنسخ مجلد ذلك المستأجر من مخزن النسخ الاحتياطي الخارجي إلى المخزن الإنتاجي. هذه العملية ينبغي أن تتم بحذر (ربما مع تعطيل وصول ذلك العميل مؤقتًا لتجنب التضارب أثناء الاستعادة).

- في حال السيناريوهات المعقدة (مثلاً عدد ضخم من الملفات حذفت)، ربما يكون الأسهل **استبدال حاوية المستأجر بالكامل** بنسخة من النسخ الاحتياطية. يمكن ذلك عن طريق حذف الحاوية الحالية (أو أرشفتها) ثم نسخ الحاوية الاحتياطية مكانها. هنا أيضًا يجب إعلام العميل بتوقف بسيط أثناء التبديل.

- **استرجاع متجهات مستأجر من Qdrant:**
إذا كان كل مستأجر له **Collection** خاص، فيمكن ببساطة استخدام **ملف لقطة ذلك الـCollection** لاستعادته. يمكن اختيار إما:

- **إنشاء مجموعة جديدة** من اللقطة (كما ذكرنا باستخدام recover API لاسم جديد) ثم تبديل التطبيق لاستخدامها بدل القديمة.
- أو **استبدال المجموعة الحالية** مباشرةً بالاستعادة (مع `priority=snapshot` لضمان الكتابة فوق البيانات الحالية) [67] [118] . هذه الخطوة ستفقد أي تغييرات حدثت بعد زمن أخذ اللقطة، فيجب إعلام العميل بذلك.
- السيناريو الأكثر أمانًا: ربما تشغيل نسخة Qdrant منفصلة واستعادة بيانات المستأجر إليها، ثم باستخدام سكربت مقارنة/تحديث بين المجموعة المستعادة والحالية – لكن هذا معقد جدًا وغالبًا غير ضروري حيث عادةً يراد إرجاع الحالة كما كانت تمامًا.

- إن كانت المجموعات مشتركة بين المستأجرين (أي مستأجرون متعددون في Collection واحدة عبر تسمية payload مثلاً)، فاستعادة واحد يعني عزل بياناته. Qdrant ليس لديه آلية تصفية داخل اللقطات، لذا قد نحتاج لاستعادة المجموعة كاملة في مكان آخر ثم أخذ فقط المتجهات التابعة لمعرف العميل ونعيد إدخالها. هذا مماثل لصعوبة الحالة في MongoDB إذا كانت مشتركة.

- **استرجاع رسائل مستأجر من RabbitMQ:**
عادة رسائل RabbitMQ لا تُسترجع بعد استهلاكها، ولكن إن افترضنا سيناريو فيه **طوابير منفصلة لكل مستأجر** ورسائل هامة تأثرت (مثلاً مستأجر طلب إعادة معالجة مجموعة رسائل لم يتسلمها بسبب عطل)، الحلول الممكنة:

- لو كانت الرسائل لا تزال في الطابور ولكن عالقة مثلاً، يمكن تحريكها (republish) أو استخراجها عبر أدوات إدارة. RabbitMQ Management UI يسمح بـ"Get Message" من الطابور يدويًا مثلًا.
- لو كانت الرسائل مفقودة تمامًا (لم تعد موجودة في النظام)، فإن الخيار هو الاعتماد على **سجل النشاط** من مستوى التطبيق لإعادة إنتاج تلك الرسائل. مثلاً قد يكون التطبيق سجل العمليات التي حصلت لكل مستأجر في قاعدة البيانات، فيمكن استخراج ما لم ينفذ وإعادة إرساله. من هنا تأتي أهمية تصميم التطبيق بتحمل.
- إذا كان هناك **لقطة من مجلد رسائل RabbitMQ** (وهي حالة نادرة كما أوضحنا)، فإن استعادة مستأجر واحد تعني استعادة **طابوره فقط**. لا توجد أداة جاهزة لدمج هذا، لكن يمكن إن حصل توقف كامل، استعادة ملفات طابور ذلك المستأجر (الموجودة في مجلد `mnesia/queuename`) إلى بيئة اختبار ثم استخدام أدوات مثل RabbitIO لإعادة إرسالها إلى النظام الجديد. هذا أشغال يدوية جدًا ونادر.

- على الأغلب، ضمانات RabbitMQ للرسائل تعتمد على عدم فقدها أولاً عبر HA، فإن ضاعت فغالبًا يتم إعلام العميل بأن بعض الرسائل فقدت ويُنصح بإعادة الإرسال من طرفه إن أمكن. لذا التركيز كما ذكرنا على الوقاية أكثر من الاستعادة.

متطلبات الاستعادة الجزئية: لكي تنجح استعادة مستأجر دون غيره، **العزل المنطقي** في التخزين أساسي. لذا أثناء تصميم قاعدة البيانات وهيكل الملفات، يؤخذ بالاعتبار ليس فقط الأمان وعزل الوصول، بل أيضًا عزل النسخ الاحتياطي. **الشفرة المتعددة المستأجرين الجيدة تجعل استخراج بيانات مستأجر مفرد أسهل كثيرًا**. مثلًا:
- قاعدة بيانات مستقلة لكل مستأجر = ملف نسخ مستقل.
- حاوية ملفات مستقلة = مجلد مستقل يمكن نسخه.
- مجموعة متجهات مستقلة = لقطة مستقلة.
- طابور أو exchange مستقل = قابل للعزل.

في حالة وجود مستأجر ضخم جدًا يطغى على البقية، ربما تعامله باستراتيجية نسخ خاصة (مثل نسخ أكثر تواتراً). أما للعملاء الآخرين نسخة مشتركة. المهم توثيق عمليات **كيفية الاستعادة الجزئية خطوة بخطوة** مسبقًا، حتى لا يُترك الفريق التقني في ارتجال لحظة الحادث. ويفضل تجربة **سيناريو استعادة زبون محدد** (Test tenant recovery scenario) كجزء من اختبارات DR للتأكد من أن الإجراءات واضحة ومجدية.

مراقبة عمليات النسخ الاحتياطي والتنبيه عند الفشل والتحقق الدوري

إن تنفيذ النسخ الاحتياطي دون نظام مراقبة يعادل عدم تنفيذه فعليًا – إذ قد تفشل عملية النسخ لأي سبب (مشكلة قرص، امتلاء مساحة، خطأ برمجي) وتفوتنا ملاحظتها، لتكتشف فقط عند الحاجة إلى الاستعادة أن آخر نسخة صالحة كانت منذ أشهر! لتفادي ذلك، من الضروري إنشاء **نظام مراقبة وإنذار** يغطي عمليات النسخ الاحتياطي جميعها:

- **التنبيه الفوري عند فشل مهمة النسخ:** كل عملية نسخ سواء عبر Cron أو أداة خارجية يجب أن تُبلغ نظامًا أو شخصًا عند فشلها. أساليب بسيطة: توجيه مخرجات السكربت إلى بريد إلكتروني للمسؤول إذا كان هناك إخفاق (يمكن فعل ذلك في Cron بسهولة). أو إرسال إشعار عبر Slack/Webhook عند انتهاء المهمة بحالة نجاح/فشل. توجد أدوات مثل **Healthchecks.io** أو **Cronitor** يمكن دمجها؛ تقوم فكرة هذه الخدمات على تلقي "نبضة" (HTTP request) من المهمة عند انتهائها بنجاح، فإن غابت النبضة يُعلم الفريق. من الجيد استخدام مثل هذه الخدمة لمراقبة مهام النسخ المهمة (مثلاً إذا لم تتصل مهمة النسخ الليلية خلال ساعة معينة يصدر تنبيه).

- **المراقبة الاستباقية للبنية التحتية للنسخ:** بالإضافة لمراقبة المهام، راقب **الموارد**:

- **مساحة التخزين الاحتياطي:** تأكد أن الخزانات التي تحفظ النسخ (محليًا أو بعيد) لا تقترب من الامتلاء. امتلاءها سيؤدي لفشل النسخ أو عدم اكتمالها. أدوات المراقبة (مثل Prometheus node exporter) يمكنها تتبع مساحة القرص وإطلاق تنبيه عند تجاوز عتبة (say 80%).
- **عرض النطاق والتوقيت:** إذا بدأت النسخ الاحتياطية تأخذ وقتًا أطول بكثير من المعتاد (ربما بسبب تضخم البيانات)، فقد تتجاوز النافذة المخصصة أو تتداخل مع وقت الذروة. لذا يجب مراقبة **مدة كل مهمة نسخ** وإنذار في حال ازدادت فجأة. يمكن ببساطة تسجيل timestamp في بداية ونهاية كل سكربت وحسابه، ثم تقرير لو كانت المدة غير اعتيادية.

- **أداء قواعد البيانات أثناء النسخ:** المراقبة قد تشمل التأكد أن MongoDB مثلًا لا يرتفع استهلاك CPU أو locking وقت النسخ إلى مستوى يضر الخدمة (وإلا قد نحتاج تعديل الجدول أو الأسلوب).

- **التحقق الدوري من صلاحية النسخ (Backup Integrity Testing):** يجب جدولة **اختبار استعادة دوري** – ليس بالضرورة لكل النسخ ولكن بشكل دوري لنوعيات مختلفة. مثلاً:

- مرة كل شهر، اختر نسخة احتياطية عشوائية قديمة لقاعدة بيانات وقم **باستعادتها في بيئة اختبار** وتحقق أن البيانات سليمة ويمكن قراءتها.
- جرب فتح بعض صور أو ملفات من النسخ الاحتياطية للتأكد أنها ليست معطوبة.
- تحقق من **سجلات hash** إن وجدت (مثلاً إذا تحتفظ بتوقيع SHA256 لكل ملف backup، تحقق منه مقابل الملف الحالي).

هذا الاختبار الدوري يكشف أي تدهور silent في وسائط التخزين أو أخطاء في عملية النسخ (كأن تكون السكربت كان ينسخ حجم صفر بايت بعد امتلاء القرص ولم ننتبه). العديد من الشركات تعتمد قاعدة أن يتم **اختبار جميع آليات الاستعادة كل 3-6 أشهر** على الأقل للتأكد من جهوزيتها [8] . كما أن CloudAMQP تنصح باختبار استعادة التعريفات التي تحتفظ بها لضمان عدم وجود "مفاجآت" [105] .

- **توثيق وإجراءات طوارئ:** يجب أن يتم توثيق خطوات استعادة البيانات بشكل واضح كما ذكرنا. وينبغي أن يكون هذا التوثيق جزءًا من **Playbook المراقبة** ؛ أي عند تلقّي إنذار بفشل نسخة، ماذا يفعل المهندس المناوب؟ يعرف مباشرةً الإجراء (كإعادة تشغيل مهمة النسخ يدويًا بعد إصلاح المشكلة، أو الاتصال بعضو أعلى). وجود هذا يزيد موثوقية النظام.

- **مراجعة خطط النسخ بشكل دوري:** لا ينبغي وضع خطة النسخ الاحتياطي ثم إهمالها. **راجع الخطة كلما تغيرت بنية النظام أو حجم البيانات بشكل ملحوظ** . ربما مع نمو عدد المستأجرين تحتاج تعديل الجدول (مثلاً تقسيم النسخ على عدة فترات لتجنب الضغط). أو مع إضافة خدمة جديدة (مثلاً لو أضفنا نظام cache أو بحث آخر) يجب دمجه في الخطة. أيضًا تقييم دوري لتكاليف التخزين مقابل الفائدة: قد تجد أنك تستطيع تمديد الاحتفاظ بنسخ أسبوعية لأكثر من شهر إذا كانت التكلفة مقبولة، وذلك مفيد للعملاء المتأخرين في اكتشاف الأخطاء.

في الختام ضمن هذا الجزء، المراقبة والاختبارات تضيفان **الثقة** في أن خطط النسخ الاحتياطي والتعافي ستعمل عند الحاجة. بدونها يكون لدينا شعور زائف بالأمان. تذكر مقولة شائعة في مجال الحماية: " نسختك الاحتياطية جيدة بقدر قدرتك على الاستعادة منها ". لذا رصد النظام والاستعداد الدائم هو ما يحول النسخ الاحتياطي من مجرد ملفات إلى **شبكة أمان حقيقية**.

أمثلة تطبيقية من أنظمة SaaS واقعية

للبرهنة على فعالية هذه الاستراتيجيات، نستعرض بعض الأمثلة الواقعية أو التقنيات المطبقة في الصناعة:

- **منصة SaaS تستخدم MongoDB وMinIO** : إحدى الحالات المفترضة هي شركة تقدم خدمة تخزين ومشاركة ملفات للعملاء (مشابهة لـDropbox ذاتيًا مثلًا) تعتمد MongoDB لإدارة البيانات الوصفية وMinIO للملفات. تتبنى هذه الشركة **نهج النسخ 3-2-1** : حيث تحتفظ بنسختين احتياطيتين من بيانات MongoDB – واحدة محليًا على خادم في مركز بيانات آخر عبر أداة مثل **Barman** (الخاصة بـPostgres ولكن MongoDB لديها ما يكافئها) أو ببساطة Cron job مع mongodump، ونسخة ثانية مُرسلة إلى تخزين سحابي (Backblaze B2) باستخدام Restic مشفر [112] . نفس الشيء لملفات MinIO: يتم تشغيل **MinIO Gateway** مرتبط بـBackblaze B2 لنسخ كل كائن جديد إليه تلقائيًا. بذلك، أي ملف يرفع من العميل يُرفع إلى موقعين (الموقع الأساسي وموقع النسخ) بشكل شبه فوري. عندما حدثت حادثة تعطل في مصفوفة التخزين الرئيسية، قاموا بتحويل الطلبات إلى مخزن B2 مؤقتًا حتى أعادوا بناء التخزين الأساسي. خسارة البيانات كانت صفر لأن التكرار الخارجي كان آنيا.

- **خدمة ذكاء اصطناعي تستخدم Qdrant** : شركة تقدم API توصيات تعتمد على Qdrant لتقارن بين تفضيلات المستخدمين. لضمان الاستمرارية، قاموا **بتفعيل نسخ متماثل replication بمقدار 3** في عنقود Qdrant – أي كل متجه مخزن على 3 عقد. كذلك طبقوا **لقطات ليلية** لكل مجموعة متجهات كبيرة [61] . هذه اللقطات تنقل تلقائيًا إلى تخزين S3 متوافق (والذي يمكن أن يكون MinIO Hybrid Gateway). في أحد المرات أدى نشر تحديث خاطئ إلى مسح بعض المجموعات، لكنهم تمكنوا من **استعادة اللقطات في**

أقل من ساعة وإعادة الخدمة. وأدركوا أهمية اختبار اللقطات دوريًا فأصبحوا يشغلون استعادة تجريبية على مجموعة واحدة كل أسبوع للتأكد من سلامة الملفات.

- **مزود RabbitMQ سحابي (CloudAMQP)** : كما استشهدنا، CloudAMQP يعتمد على **نسخ التعريفات كل 8 ساعات** كخدمة مجانية [79] ، لكنه لا يقدم نسخًا للرسائل. بدلاً من ذلك، لديهم عروض **للتحول إلى Cluster آخر** (DR cluster) إما بارد أو ساخن حسب خطة العميل [100] [97] . العديد من المؤسسات المالية التي تستخدم RabbitMQ داخليًا تتبع نفس المبدأ: تشغيل عنقودين في مواقع مختلفة مع **Federation لجزء من الرسائل المهمة** لضمان توفر نسخة ثانية آنية [98] . على سبيل المثال، شركة تداول لديها مسار أوامر عبر RabbitMQ، قامت بإعداد Exchange Federation لعنقود DR حيث كل أمر يُرسل ينسخ فورًا، فلو تعطل الموقع الأساسي، تنتقل المعالجة إلى الموقع الثانوي بدون فقدان أي أمر. طبعاً هذا تصميم معقد لكنه يُستخدم في الحالات الحرجة.

- **استخدام الأدوات المفتوحة المصدر في الشركات الصغيرة:** كثير من الشركات الناشئة الصغيرة التي تعمل على VPS استخدمت **BorgBackup** لتحقيق نسخ مشفرة تلقائية إلى خوادم أخرى. يذكر أحد التقارير أن شركة تقنية قامت بدمج Borg مع **Cron** و**Slack Webhook** بحيث أي نجاح أو فشل نسخة يتم إشعار فريق التطوير مباشرة. كما قاموا بتخزين أرشيف Borg في خادم لدى أحد المؤسسين كنسخة خارجية إضافية (تنفيذ فعلي لوجود نسخة Off-site). عندما تعرضت الشركة لهجوم فدية شلّ خوادم الإنتاج، تمكنوا من إعادة البناء خلال ساعات قليلة من أرشيف Borg الموجود خارجيًا ولم يدفعوا الفدية، مؤكّدين بذلك قيمة الاستثمار المسبق في النسخ الاحتياطي.

- **منصة Kubernetes SaaS** : بعض المنصات التي تستضيف تطبيقات SaaS على Kubernetes تستخدم **Velero** (أداة نسخ احتياطي لحاويات Kubernetes) لنسخ قواعد البيانات وحجوم التخزين عبر تكاملها مع S3. مثلاً شركة تستخدم MongoDB وMinIO داخل K8s، اعتمدت Velero لعمل نسخ مجدولة للتخزين الدائم PVC (الذي يشمل MongoDB datafiles وMinIO buckets) وإرسالها إلى مخزن S3 مركزي. هذا وفر لهم حلًا موحدًا قابل للتوسع مع إضافة عملاء جدد – كل ما عليهم هو تعريف سياسة Velero جديدة عند إضافة تطبيق جديد. كما أن Velero ينسّق استعادة جميع الأجزاء معًا مما يسهل عملية DR الشاملة. ورغم أننا في VPS بدون Kubernetes، إلا أن فكرة وجود منسق *Backup* يمكن أن تطبق باستخدام سكربت شامل أو أدوات مثل Ansible لأتمتة أخذ واستعادة نسخ متعدد المكونات بترتيب محدد.

هذه الأمثلة وغيرها تؤكد أن **النسخ الاحتياطي والتعافي من الكوارث ليس مجرد نظرية، بل ممارسة حيوية** في تشغيل الأنظمة متعددة المستأجرين. الشركات التي تستثمر فيه وتحسّن خططه باستمرار هي الأقدر على تجنب كوارث فقدان البيانات التي قد تكون مدمرة لسمعتها وأعمالها. بتطبيق الاستراتيجيات المذكورة ومراجعتها دوريًا، يمكن لأي منصة SaaS – حتى على موارد محدودة في VPS – أن تحقق مستوى عالٍ من الأمان والجاهزية أمام أسوأ السيناريوهات.

المصادر: تم الاستناد في إعداد هذا التقرير إلى وثائق رسمية وتقارير حديثة حول MongoDB وMinIO وQdrant وRabbitMQ وغيرها [21] [116] [49] [81] ، لضمان حداثة وصحة التوصيات الواردة.

---

[1] [8] [81] [82] [84] [85] [86] [87] [88] [89] [96] Effective Disaster Recovery Strategies for RabbitMQ | Reintech media
https://reintech.io/blog/effective-disaster-recovery-strategies-rabbitmq

[2] [5] [6] [7] [19] [25] [27] [29] [30] [31] [116] MongoDB Backup: Essential Strategies to Safeguard Your Data
https://trilio.io/resources/mongodb-backup/

[3] [4] Backup Strategies: Why the 3-2-1 Backup Strategy is the Best
https://www.backblaze.com/blog/the-3-2-1-backup-strategy/

The New Math on Backup and Replication [115] [53] [46] [45] [44] [39] [38] [37] [36] [35] [34] [20] [13] [12] [11] [10] [9]
/https://blog.min.io/the-new-math-on-backup-and-replication

Full vs. Incremental vs. Differential Backups: Comparing Backup Types [18] [17] [16] [15] [14]
/https://www.percona.com/blog/what-are-full-incremental-and-differential-backups

- Backup Methods for a Self-Managed Deployment - Database Manual MongoDB Docs [117] [28] [26] [24] [23] [21]
/https://www.mongodb.com/docs/manual/core/backups

Performing an Incremental Backup for a MongoDB Cluster [113] [22]
https://documentation.commvault.com/v11/software/performing_incremental_backup_for_mongodb_cluster.html

Protecting Software-defined Object Storage With MinIO's Replication Best Practices | Volito [52] [51] [49] [48] [47] [43] [42] [41] [40] [33] [32]
/https://volito.digital/protecting-software-defined-object-storage-with-minios-replication-best-practices

Server-Side Encryption with Client-Managed Keys (SSE-C) - MinIO [50]
https://min.io/docs/minio/kubernetes/eks/administration/server-side-encryption/server-side-encryption-sse-c.html

Snapshots - Qdrant [118] [70] [69] [68] [67] [66] [65] [64] [63] [60] [59] [58] [57] [56] [55] [54]
/https://qdrant.tech/documentation/concepts/snapshots

Backup Clusters - Qdrant [76] [75] [74] [73] [72] [71] [62] [61]
/https://qdrant.tech/documentation/cloud/backups

Backup and Restore | RabbitMQ [95] [94] [93] [92] [91] [83] [78] [77]
https://www.rabbitmq.com/docs/backup

- Disaster recovery strategies for your CloudAMQP cluster CloudAMQP [105] [104] [103] [102] [101] [100] [99] [98] [97] [80] [79]
https://www.cloudamqp.com/blog/disaster-recovery-strategies-for-your-cloudamqp-cluster.html

RabbitIO: A Tool to Backup and Restore Messages from RabbitMQ [90]
/https://underthehood.meltwater.com/blog/2018/08/03/rabbitio-a-tool-to-backup-and-restore-messages-from-rabbitmq

Best Open Source Tools for Backup and Restore | simplyblock [114] [112] [111] [110] [109] [108] [107] [106]
/https://www.simplyblock.io/blog/open-source-tools-for-backup-and-restore